BORLAND LANGUAGES HELP COMPILER

BORLAND

Help Compiler

BORLAND INTERNATIONAL, INC. 100 BORLAND WAY
P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001

Copyright © 1991 by Borland International. All rights reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

PRINTED IN THE USA
10 9 8 7 6 5 4 3

R1

Table of Contents

Chapter 1:
Providing Help: The Help System

Creating a Help System:
The Development Cycle 1
How Help Appears to the User 4
How Help Appears to the Help Writer 5
How Help Appears to the Help Programmer 6
Summary 6

Chapter 2:
Planning the Help System

Developing a Plan 7
Defining the Audience 8
Planning the Content of the Help System 8
Planning the Structure of Help Topics 9
Displaying Context-Sensitive Help Topics 11
Determining the Topic File Structure . 13
Choosing a File Structure for Your Application 14
Designing the Appearance of Help Topics 16
Layout of the Help Text 16
Type Fonts and Sizes 18
Graphic Images 20
Summary 21

Chapter 3:
Creating the Help Topic Files

Choosing an Authoring Tool 23
Structuring Help Topic Files 24
Coding Help Topic Files 25
Assigning Build Tags 26
Assigning Context Strings 28
Assigning Titles 29
Assigning Keywords 30
Creating Multiple Key-Word Tables 31
Assigning Browse Sequence Numbers 32
Organizing Browse Sequences . . 32
Coding Browse Sequences 34
Creating Cross-References Between Topics 34
Defining Terms 35
Creating Definition Topics 35
Coding Definitions 36
Inserting Graphic Images 37
Creating and Capturing Bitmaps . . 37
Placing Bitmaps Using a Graphical Word Processor 38
Placing Bitmaps by Reference 38
Managing Topic Files 39
Keeping Track of Files and Topics . 40
Creating a Help Tracker 40
Summary 42

Chapter 4:
Building the Help File

Creating the Help Project File 43
Specifying Topic Files:
The Files Section 44
Specifying Build Tags:
The BuildTags Section 45
Specifying Options:
The Options Section 46
Specifying Error Reporting:
The Warning Option 46
Specifying Build Topics:
The Build Option 47
Specifying the Root Directory:
The Root Option 48
Specifying the Index:
The Index Option 49
Assigning a Title to the Help System:
The Title Option 50

Converting Fonts:
The Forcefont Option 50
Changing Font Sizes:
The Mapfontsize Option 51
Multiple Keyword Tables:
The Multikey Option 51
Compressing the File:
The Compress Option 52
Specifying Alternate Context Strings:
The Alias Section 53
Mapping Context-Sensitive Topics:
The Map Section 54
Including Bitmaps by Reference:
The Bitmaps Section 56
Compiling Help Files 56
Using the Help Compiler 57
Programming the Application to
Access Help 58
Calling WinHelp from an
Application 58
Getting Context-Sensitive Help . . . 60
Shift+F1 Support 60
F1 Support 63
Getting Help on an Item Listed on
the Help Menu 66
Accessing Additional Key-Word
Tables 66
Canceling Help 67
Summary 68

Chapter 5:
Help Examples and Compiler Error Messages

Help Topic Examples 70
Help Compiler Error Messages 75
Errors During Processing of Project
File 77
Errors During Processing of RTF Topic
Files 81

Index 87

CHAPTER 1

Providing Help: The Help System

A Help system provides users with online information about an application. Creating the system requires the efforts of both a Help writer and a Help programmer. The Help writer plans, writes, codes, builds, and keeps track of Help topic files, which are text files that describe various aspects of the application. The Help programmer ensures that the Help system works properly with the application.

This chapter describes the following topics:

- Creating the Help system
- How Help appears to the user
- How Help appears to the Help writer
- How Help appears to the Help programmer

This chapter and those that follow assume you are familiar with Microsoft Windows Help. The chapters use examples from sample applications (written in C and Pascal) provided on your disks. If you are unfamiliar with Windows Help, take a moment to run the sample application and use Helpex Help.

Creating a Help System: The Development Cycle

The creation of a Help system for a Windows application comprises the following major tasks:

1. Gathering information for the Help topics.
2. Planning the Help system.

Chapter 2, "Planning the Help System," describes considerations you should keep in mind when planning your Help system.

3. Writing the text for the Help topics.
4. Entering all required control codes into the text files.

   Control codes determine how the user can move around the Help system. The section titled, "How Help Appears to the Writer," includes an example of several control codes. Chapter 3, "Creating the Help Topic Files," describes the codes in detail.

5. Creating a project file for the build.

   The Help project file provides information that the Help Compiler needs to build a Help resource file. Chapter 4, "Building the Help File," describes the Help project file.

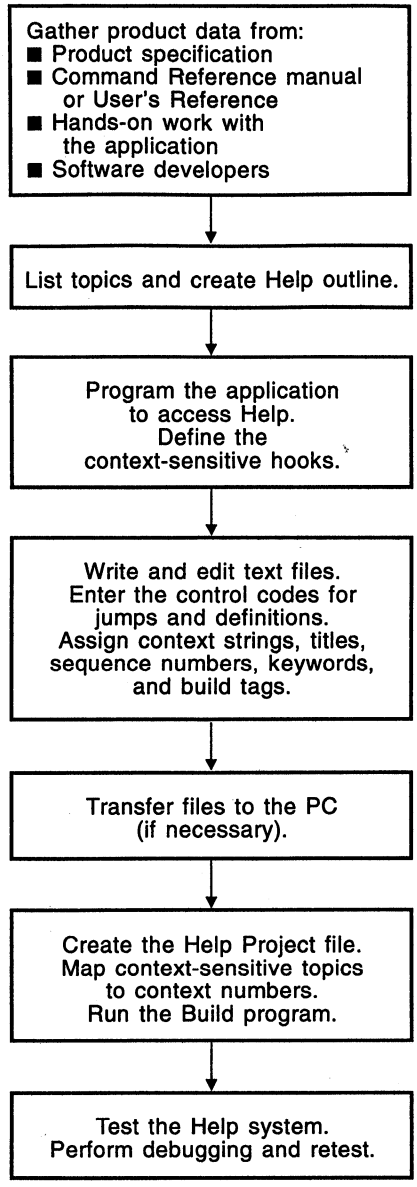
6. Building the Help resource file.

   The Help resource file is a compiled version of the topic files the writer creates. Chapter 4, "Building the Help File," describes how to compile a Help resource file.

7. Testing and debugging the Help system.
8. Programming the application so that it can access Windows Help.

The following flow diagram shows the general work flow in the conception and development of the Help system.

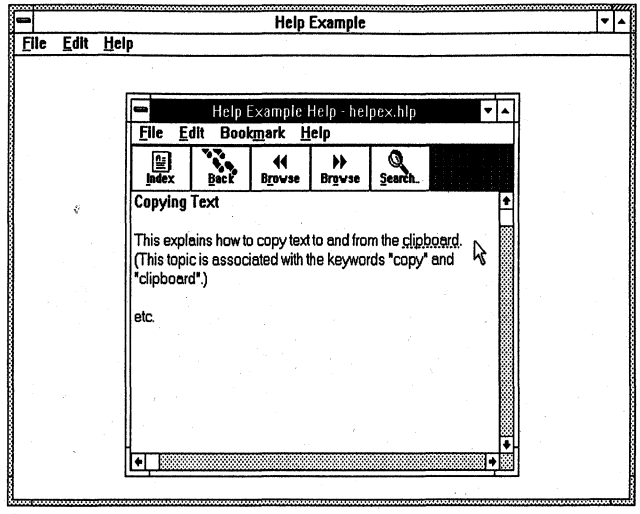
Figure 1.1
Work flow diagram


Gather product data from:
■ Product specification
■ Command Reference manual or User's Reference
■ Hands-on work with the application
■ Software developers
List topics and create Help outline.
Program the application to access Help. Define the context-sensitive hooks.
Write and edit text files. Enter the control codes for jumps and definitions. Assign context strings, titles, sequence numbers, keywords, and build tags.
Transfer files to the PC (if necessary).
Create the Help Project file. Map context-sensitive topics to context numbers. Run the Build program.
Test the Help system. Perform debugging and retest.

How Help Appears to the User

To the user, the Help system appears to be part of the application, and is made up of text and graphics displayed in the Help window in front of the application. Figure 1.2 illustrates the Help window that appears when the user asks for help with copying text in Helpex.

Figure 1.2
Helpex help window



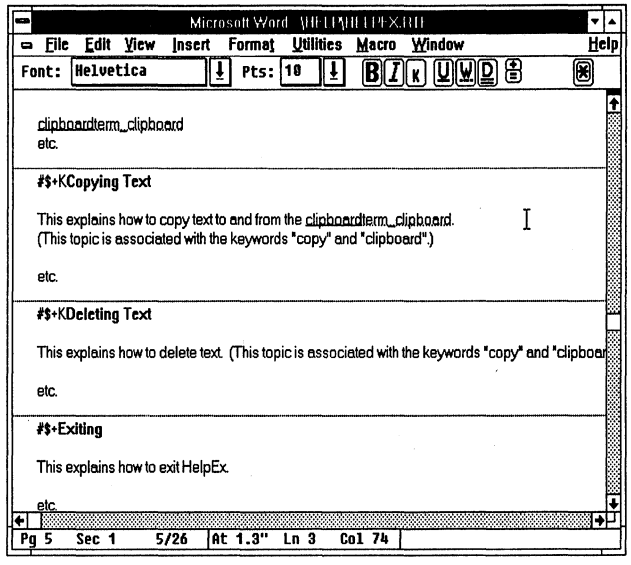
The Help window displays one sample Help topic, a partial description of how to perform one task. In Figure 1.2, the first sentence includes a definition of the word "clipboard." By pressing the mouse button when the cursor is on the word (denoted by dotted underlined text), the user can read the definition, which appears in an overlapping box as long as the mouse button is held down.

Cross-references to related topics are called jumps. By clicking on a jump term for a related topic (denoted by underlined text), the user changes the content of the Help window to a description of the new topic or command. Figure 1.2 includes a look-up to the definition of "clipboard."

How Help Appears to the Help Writer

To the writer, the Help system is a group of topic files, which are text files that include special codes. Figure 1.3 illustrates the source text that corresponds to the topic shown in Figure 1.2.

Figure 1.3
Topic file


Microsoft Word
File Edit View Insert Format Utilities Macro Window Help
Font: Helvetica Pts: 10
clipboardterm_clipboard
etc.
#$+KCopying Text
This explains how to copy text to and from the clipboardterm_clipboard.
(This topic is associated with the keywords "copy" and "clipboard".)
etc.
#$+KDeleting Text
This explains how to delete text. (This topic is associated with the keywords "copy" and "clipboar
etc.
#$+Exiting
This explains how to exit HelpEx.
etc.
Pg 5 Sec 1 5/26 At 1.3" Ln 3 Col 74

To create this topic, the Help writer describes the task, formats the text, and inserts codes using strikethrough text, underlined text, and footnotes. In place of strikethrough, the writer can use double underlining if the word processor does not support strikethrough formatting. Footnotes in the text contain linking information required by the Help Compiler. Chapter 2, "Planning the Help System," discusses formatting considerations. Chapter 3, "Creating the Help Topic Files," describes how to create topics and enter the special codes that the Help system uses.

How Help Appears to the Help Programmer

To the programmer, Windows Help is a stand-alone Windows application which the user can run like any other application. Your application can call the **WinHelp** function to ask Windows to run the Help application and specify which topic to display in the Help window.

See Chapter 4, "Building the Help File," for details about the Help application programming interface (API).

Summary

The Help system is made up of topics linked via hypertext. The topics and links appear differently on the screen to the user than they do in a topic file to the writer. To the programmer, Help is a stand-alone application.

For more information about related topics, see the following:

Topic	Reference
Coding Help topics	*Tools*: Chapter 3, "Creating the Help Topic Files"
Compiling Help files	*Tools*: Chapter 4, "Building the Help File"

CHAPTER 2

Planning the Help System

The initial task for the Help writer is to develop a plan for creating the system. This chapter discusses planning the Help system for a particular application.

The chapter covers the following topics:

- Developing a plan
- Determining the topic file structure
- Designing the visual appearance of Help topics

Developing a Plan

Before you begin writing Help topics using the information you have gathered, you and the other members of the Help team should develop a plan that addresses the following issues:

- The audience for your application
- The content of the Help topics
- The structure of topics
- The use of context-sensitive topics

You might want to present your plan in a design document that includes an outline of Help information, a diagram of the structure of topics, and samples of the various kinds of topics your system will include. Keep in mind that context-sensitive Help requires increased development time, especially for the application programmer.

Defining the Audience

The audience you address determines what kind of information you make available in your Help system and how you present the information.

Users of Help systems might be classified as follows:

User	Background
Computer novice	Completely new to computing.
Application novice	Some knowledge of computing, but new to your kind of application. For example, if you are providing Help for a spreadsheet program, the application novice might be familiar only with word-processing packages.
Application intermediate	Knowledgeable about your kind of application.
Application expert	Experienced extensively with your type of application.

Keep in mind that one user may have various levels of knowledge. For example, the expert in word processors may have no experience using spreadsheets.

Planning the Content of the Help System

You should create topics that are numerous enough and specific enough to provide your users with the help they need.

Novice users need help learning tasks and more definitions of terms. More sophisticated users occasionally seek help with a procedure or term, but most often refresh their memory of commands and functions. The expert user tends only to seek help with command or function syntax, keyboard equivalents, and shortcut keys.

There are no rules for determining the overall content of your Help system. If you are providing Help for all types of users, you will want to document commands, procedures, definitions, features, functions, and other relevant aspects of your application. If you are providing help for expert users only, you might want to omit topics that describe procedures. Let your audience definition guide you when deciding what topics to include.

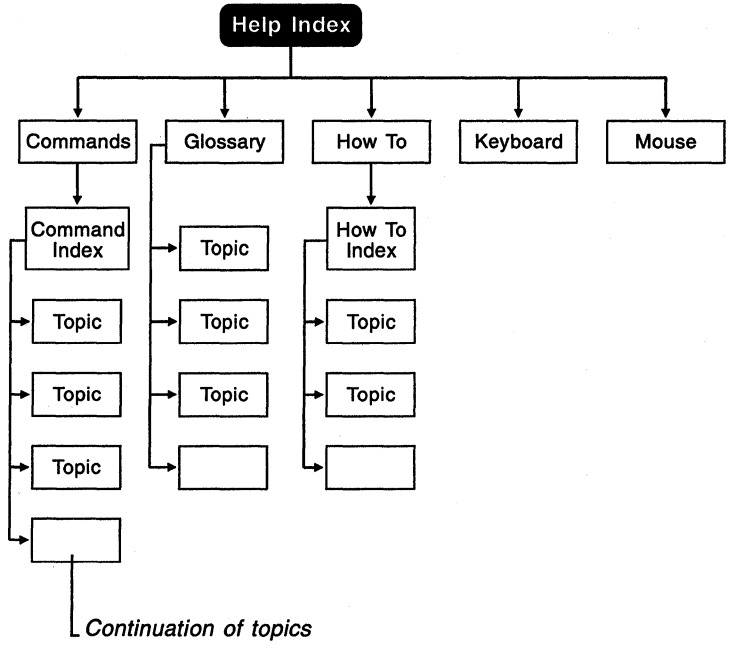
Keep in mind that the decision to implement context-sensitive Help is an important one. Context-sensitive Help demands a close working relationship between the Help author and the application programmer, and will therefore increase the development time necessary to create a successful Help system.

Planning the Structure of Help Topics

Many Help systems structure topics hierarchically. At the top of the hierarchy is an index or a table of contents, or both. The index and table of contents list individual topics or categories of topics available to the user.

Topics themselves can be related hierarchically. Each successive step takes the user one level down in the hierarchy of the Help system until the user reaches topic information. The hierarchical relationship of Help topics determines in part how the user navigates through the Help system. Figure 2.1 illustrates a possible hierarchy:

Figure 2.1
Example of a help hierarchy



Helpex contains an index that lists several categories of topics. Each category includes a secondary index, which lists topics in the category, and the topics themselves.

Moving from the index to a topic, the user goes from the general to the specific.

The hierarchical structure provides the user a point of reference within Help. Users are not constrained to navigate up and down the hierarchy; they can jump from one topic to another, moving across categories of topics. The effect of jumps is to obscure hierarchical relationships. For example, the Windows Help application contains a search feature that lets the user enter a key word into a dialog box and search for topics associated with that key word. The Help application then displays a list of titles to choose from in order to access information that relates to the key word.

Because users often know which feature they want help with, they can usually find what they want faster using the search feature than they can by moving through the hierarchical structure. For more information about the search feature, see the section in Chapter 3 titled, "Assigning Key Words."

In addition to ordering topics hierarchically, you can order them in a logical sequence that suits your audience. The logical sequence, or "browse sequence," lets the user choose the Browse button to move from topic to topic. Browse sequences are especially important for users who like to read several related topics at once, such as the topics covering the commands on the File menu. For more information about browse sequences, see Chapter 3, "Creating the Help Topic Files."

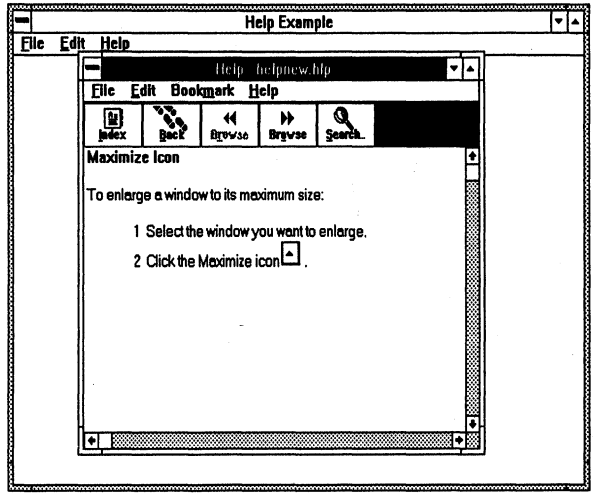
Whichever structure you decide to use, try to minimize the number of lists that users must traverse in order to obtain information. Also, avoid making users move through many levels to reach a topic. Most Help systems function quite well with only two or three levels.

Displaying Context-Sensitive Help Topics

Windows Help supports context-sensitive Help. When written in conjunction with programming of the application, context-sensitive Help lets the user press F1 in an open menu to get help with the selected menu item. Alternatively, the user can press SHIFT+F1 and then click on a screen region or command to get help on that item.

For example, if the user presses SHIFT+F1, then clicks on the maximize icon when using the sample application Helpex, the Help system displays the information illustrated in Figure 2.2:

Figure 2.2
Help example maximize icon topic



Developing context-sensitive Help requires coordination between the Help writer and the application programmer so that Help and the application pass the correct information back and forth.

To plan for context-sensitive Help, the Help author and the application programmer should agree on a list of context numbers. Context numbers are arbitrary numbers that correspond to each menu command or screen region in the application, and are used to create the links to the corresponding Help topics. You can then enter these numbers, along with their corresponding context-string identifiers, in the Help project file, which the Help Compiler uses to build a Help resource file. The section in Chapter 4 titled "Creating the Help Project File" provides details on how to create a Help project file.

The context numbers assigned in the Help project file must correspond to the context numbers that the application sends at run time

to invoke a particular topic. See the section in Chapter 4 titled "Programming the Application to Access Help" for more information on assigning context numbers.

If you do not explicitly assign context numbers to topics, the Help Compiler generates default values by converting topic context strings into context numbers. See the section in Chapter 4 titled "Mapping Context-Sensitive Topics: The Map Section" for more information on context-sensitive Help and context strings.

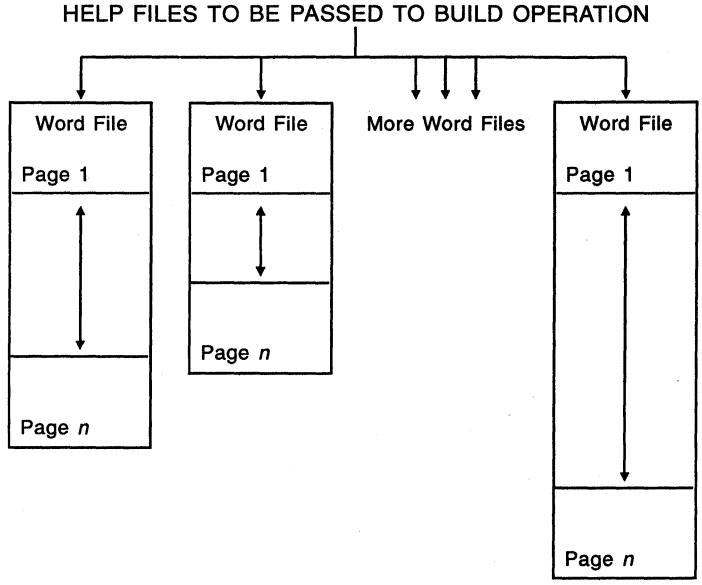
To manage context numbers and file information, you might want to create a Help tracker to list the context numbers for your context-sensitive topics. See the section in Chapter 3 titled "Creating the Help Tracker" for information about using a tracker.

Determining the Topic File Structure

The Help file structure remains essentially the same for all applications even though the context and number of topic files differ. Topic files are segmented into the different topics by means of page breaks. When you build the Help system, the compiler uses these topic files to create the information displayed for the user in the application's Help window.

Figure 2.3 shows this basic file structure.

Figure 2.3
Basic help file structure



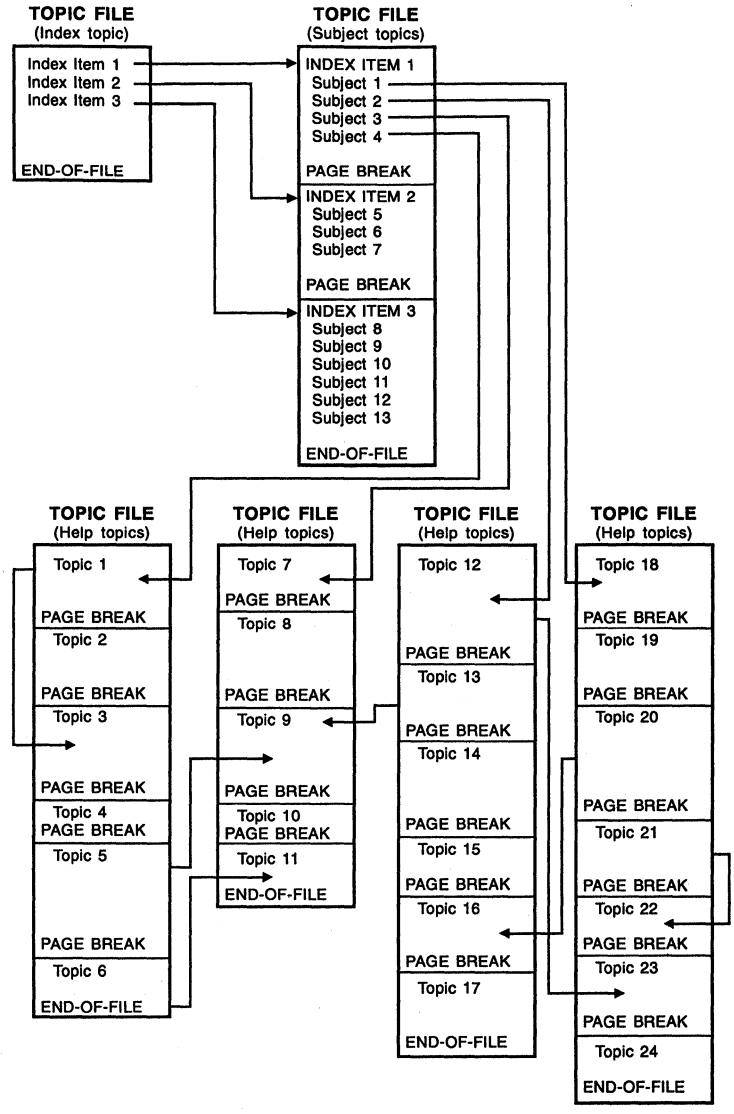
Choosing a File Structure for Your Application

When choosing a file structure for your Help system, consider the scope and content of the Help system you are planning. For example, you could place all Help topics in a single large topic file. Or, you could place each Help topic in a separate file. Neither of these file structures is generally acceptable. An enormous single file or too many individual files can present difficulties during the creation of the Help resource file.

The number of topics relates to the number of features covered by the Help system. Consequently, you cannot make extensive changes to one without making changes to the other. For instance, if a number of additional product features are added to Help, then additional topics must be created to accommodate the new information.

Figure 2.4 illustrates the file structure of a possible Help system. The number of topics and topic files is limited to simplify the diagram and more clearly show the concept of linking the topics together through jumps, shown in the figure as arrows. The figure is not intended to show the number of files that can be included in the Help file system. Moreover, the figure does not show how topic files are ordered using the browse feature.

Figure 2.4
Help file structure showing hypertext jumps



Designing the Appearance of Help Topics

How the information in the Help window appears to the user is primarily a function of the layout of the Help topic. The Windows Help application supports a number of text attributes and graphic images you can use to design your Help window.

This section provides general guidelines for designing a window and describes fonts and graphic images that Windows Help supports.

Layout of the Help Text

Help text files are not limited to plain, unformatted text. You can use different fonts and point sizes, include color and graphics to emphasize points, indent paragraphs to present complex information, and use a variety of other visual devices to present your information.

Research on screen format and Help systems has produced general guidelines for presenting information to users. The summary of this chapter lists some sources of this research. The following list summarizes the findings of these studies.

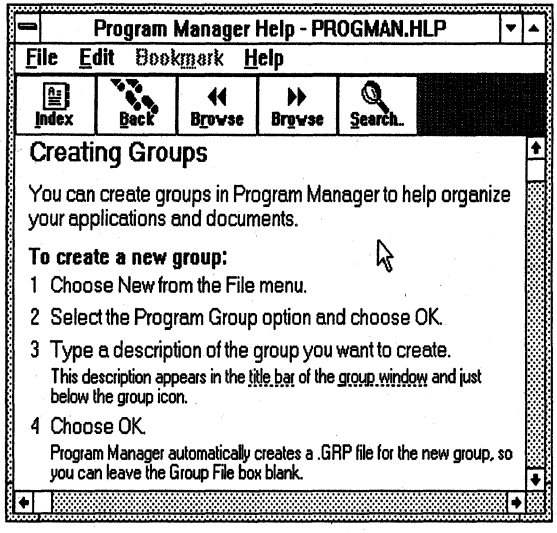
Design Issue	Guideline
Language	Use language appropriate for the audience you have defined. Language that is too sophisticated for your audience can frustrate users by requiring them to learn the definition of unfamiliar terms and concepts.
Amount of text	Use a minimum of text. Studies indicate that reading speed decreases by 30 percent when users read online text rather than printed text. Minimal, concise text helps users compensate for the decreased reading speed.
Paragraph length	Use short paragraphs.

Design Issue	Guideline
	Online users become overloaded with text more easily than do readers of printed material. Breaking your text into short paragraphs helps avoid this problem.
White space	Use it to help group information visually. White space is important to making online text more readable. Use it liberally, while also considering the overall size that a topic will occupy on the screen. Users tend to think there is more information on a screen than exists. For example, if the ratio of white space to text is 50:50, users perceive the ratio to be 40:60.
Highlighting	Use highlighting techniques judiciously. Windows Help provides a variety of highlighting devices, such as font sizes, font types, and color. Using a few devices can help users find information quickly. Using many devices will decrease the effectiveness of your visual presentation and frustrate users. As with print-based documentation, use only one or two fonts at a time.
Graphics and icons	Use graphics to support the explanation of visual events. Windows Help supports the use of bitmapped graphic images. Use appropriate images to help explain the function of icons and screen elements in your application. Remember that graphics will draw the user's eye before the accompanying text. Be sure to crop your images to remove distracting information. Use color images only if you are certain that all your users' systems have color capability. As with context-sensitive Help, consider the additional time necessary to create accurate and meaningful graphic images.

Design Issue	Guideline
Design consistency	Be rigorously consistent in your design. Users expect the appearance of Help topics to be the same, regardless of the information presented. Consistent titling, highlighting, fonts, and positioning of text in the window is essential to an effective Help system.

Figure 2.5 illustrates a Help window that follows these design principles.

Figure 2.5
Layout of help text


Program Manager Help - PROGMAN.HLP
File Edit Bookmark Help
Index
Back
Browse
Browse
Search..
Creating Groups
You can create groups in Program Manager to help organize your applications and documents.
To create a new group:
1 Choose New from the File menu.
2 Select the Program Group option and choose OK.
3 Type a description of the group you want to create.
This description appears in the title bar of the group window and just below the group icon.
4 Choose OK.
Program Manager automatically creates a .GRP file for the new group, so you can leave the Group File box blank.

Type Fonts and Sizes

The Windows Help application can display text in any font and size available to the system. When the topic files are passed to the build process, the Help Compiler attempts to use the fonts and sizes found in the topic files. If a font or point size cannot be matched exactly when the Help file is displayed by Windows Help, the closest available font and size on the user's system will be used.

Windows ships with only certain fonts in specific font sizes. If you write Help files using these fonts and sizes, the displayed Help file will closely match the printed word-processed file. Because fonts other than those shipped with Windows may not be available on

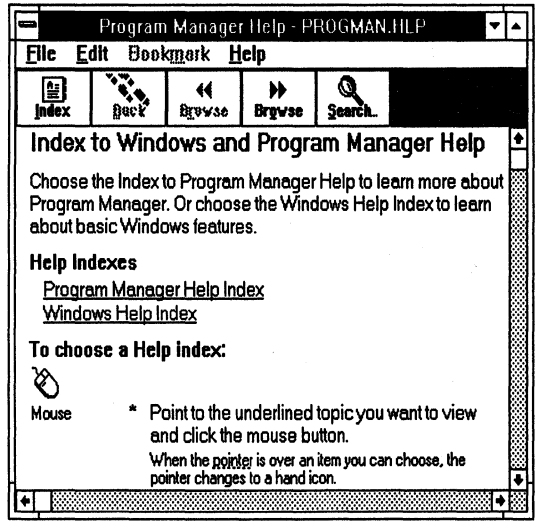
users' machines, you might want to limit your font selection to the shipped Windows fonts.

The fonts included with Windows are:

- Courier 10,12,15
- Helv 8,10,12,14,18,24
- Modern
- Roman
- Script
- Symbol 8,10,12,14,18,24
- Tms Rmn 8,10,12,14,18,24

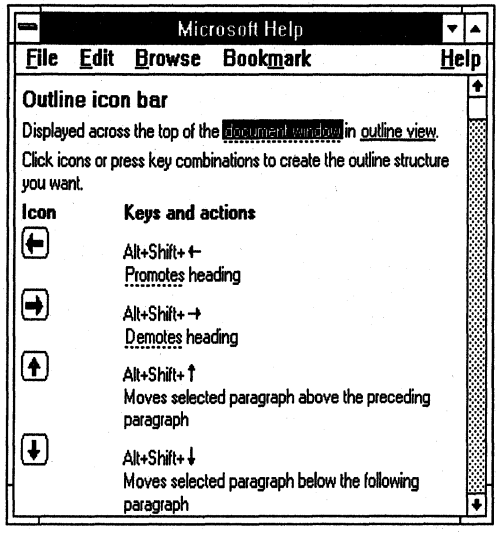
Figure 2.6 illustrates a Help window with Helvetica-font text:

Figure 2.6
Helvetica font


Program Manager Help - PROGMAN.HLP
File Edit Bookmark Help
Index
Back
Browse
Browse
Search...
Index to Windows and Program Manager Help
Choose the Index to Program Manager Help to learn more about Program Manager. Or choose the Windows Help Index to learn about basic Windows features.
Help Indexes
Program Manager Help Index
Windows Help Index
To choose a Help index:
Mouse
* Point to the underlined topic you want to view and click the mouse button.
When the pointer is over an item you can choose, the pointer changes to a hand icon.

Since Windows Help supports any Windows font, special symbols such as arrows can be included in your topics by using the Symbol font, as shown in Figure 2.7:

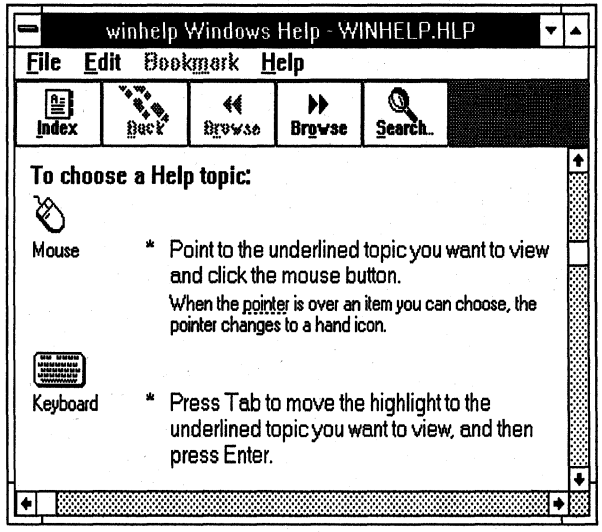
Figure 2.7
Symbol font


Microsoft Help
File Edit Browse Bookmark Help
Outline icon bar
Displayed across the top of the document window in outline view.
Click icons or press key combinations to create the outline structure you want.
Icon
Keys and actions
Alt+Shift+←
Promotes heading
Alt+Shift+→
Demotes heading
Alt+Shift+↑
Moves selected paragraph above the preceding paragraph
Alt+Shift+↓
Moves selected paragraph below the following paragraph

Graphic Images

The Windows Help application allows you to embed graphics in the Help file. Graphics can be placed and displayed anywhere on the page. Text can appear next to the graphic, as shown in Figure 2.8:

Figure 2.8
Help graphics


winhelp Windows Help - WINHELP.HLP
File Edit Bookmark Help
Index
Back
Browse
Browse
Search...
To choose a Help topic:
Mouse
* Point to the underlined topic you want to view and click the mouse button.
When the pointer is over an item you can choose, the pointer changes to a hand icon.
Keyboard
* Press Tab to move the highlight to the underlined topic you want to view, and then press Enter.

Color graphic images can be included, provided you use only the available Windows system colors. If you use graphics tools that

support an enhanced color palette to create or capture images, these images may not always display with the intended colors. And since you cannot control the color capabilities on a user's machine, you might want to limit your graphic images to black and white bitmaps.

Keep in mind that graphics are most effective when they contribute to the learning process. Graphics not tied to the information are usually distracting rather than helpful and should be avoided. See the section in Chapter 3 titled "Inserting Graphic Images" for more information on placing graphics into your Help files.

Summary

This chapter described how to plan a Help system, including defining the audience, planning the content, implementing context-sensitive Help, structuring topic files, and designing the layout of Help topics.

For additional information about planning your Help system, see the following:

Topic	Reference
Creating topic files	*Tools*: Chapter 3, "Creating the Help Topic Files"
Screen design	Bradford, Annette Norris. "Conceptual Differences Between the Display Screen and the Printed Page." *Technical Communication* (Third Quarter 1984): 13-16
	Galitz, Wilbert O. *Handbook of Screen Format Design.* 3d ed. Wellesley, MA: QED Information Sciences, Inc., 1989
	Houghton, Raymond C., Jr. "Online Help Systems: A Conspectus." *Communications of the ACM* 27(February 1984): 126-133
	Queipo, Larry. "User Expectations of Online Information." *IEEE Transactions on Professional Communications* PC 29 (December 1986): 11-15

Creating the Help Topic Files

Probably the most time-consuming task in developing a Help system for your application is creating the Help topic files from which you compile the Help system. Help topic files are text files that define what the user sees when using the Help system. The topic files can define various kinds of information, such as an index to information on the system, a list of commands, or a description of how to perform a task.

Creating topic files entails writing the text that the user sees when using Help, and entering control codes that determine how the user can move from one topic to another.

This chapter describes the following topics:

- Choosing an authoring tool
- Structuring Help topic files
- Coding Help topic files
- Managing Help topic files

Choosing an Authoring Tool

To write your text files, you will need a Rich Text Format (RTF) editor, which lets you create the footnotes, underlined text, and strikethrough or double-underlined text that indicate the control codes. These codes are described in the section "Coding Help Topic Files." Your choices include, but are not limited to:

- Microsoft Word for Windows, version 1.0.
- Microsoft Word for the PC, version 5.0.

- Microsoft Word for the Macintosh, version 3.0 or 4.0.
- Other word processors that support RTF.

Microsoft Word's RTF capability allows you to insert the coded text required to define Help terms, such as jumps, keywords, and definitions. If you choose an editor other than one of the Microsoft Word products, make sure it will create Help files that work as you intend.

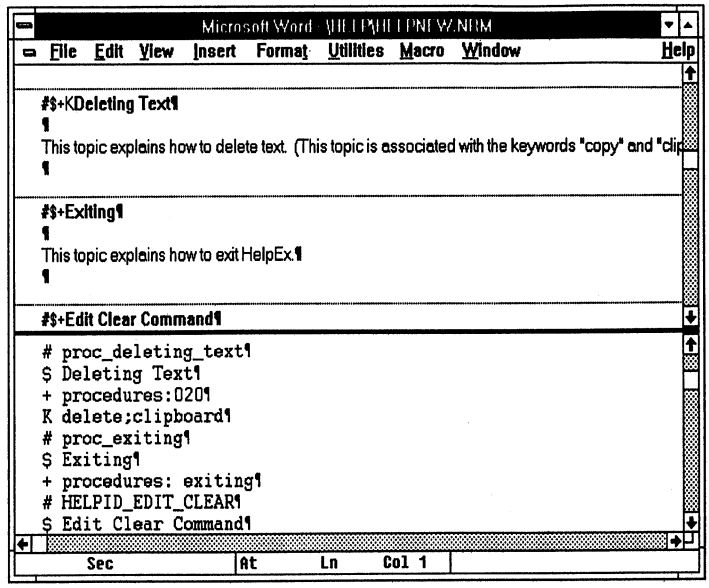
Structuring Help Topic Files

A Help topic file typically contains multiple Help topics. To identify each topic within a file:

- Topics are separated by hard page breaks.
- Each topic accessible via a hypertext link must have a unique identifier, or context string.
- Each topic can have a title.
- Each topic can have a list of keywords associated with it.
- Each topic can have a build-tag indicator.
- Any topic can have an assigned browse sequence.

Figure 3.1 illustrates part of the topic file that contains descriptions of how to perform tasks using the sample application Helpex.

Figure 3.1
Sample help topics



For information about inserting page breaks between topics, see the documentation for the editor you are using. For information about assigning context strings and titles to topics, see the following sections.

Coding Help Topic Files

The Help system uses control codes for particular purposes:

Control Code	Purpose
Asterisk (*) footnote	Build tag—Defines a tag that specifies topics the compiler conditionally builds into the system. Build tags are optional, but they must appear first in a topic when they are used.
Pound sign (#)	Context string—Defines a context string that uniquely identifies a topic. Because hypertext relies on links provided by context strings, topics without context strings can only be accessed using keywords or browse sequences.

Control Code	Purpose
Dollar sign ($) footnote	Title—Defines the title of a topic. Titles are optional.
Letter "K" footnote	Keyword—Defines a keyword the user uses to search for a topic. Keywords are optional.
Plus sign (+) footnote	Browse sequence number—Defines a sequence that determines the order in which the user can browse through topics. Browse sequences are optional. However, if you omit browse sequences, the Help window will still include the Browse buttons, but they will be grayed.
Strikethrough or double-underlined text	Cross-reference—Indicates the text the user can choose to jump to another topic.
Underlined text	Definition—Specifies that a temporary or "look-up" box be displayed when the user holds down the mouse button or ENTER key. The box can include such information as the definition of a word or phrase, or a hint about a procedure.
Hidden text	Cross-reference context string—Specifies the context string for the topic that will be displayed when the user chooses the text that immediately precedes it.

If you are using build tags, footnote them at the very beginning of the topic. Place other footnotes in any order you want. For information about assigning specific control codes, see the following sections.

Assigning Build Tags

Build tags are strings that you assign to a topic in order to conditionally include or exclude that topic from a build. Each topic can have one or more build tags. Build tags are not a necessary component of your Help system. However, they do provide a means of supporting different versions of a Help system without having to create different source files for each version. Topics without build tags are always included in a build.

You insert build tags as footnotes using the asterisk (*). When you assign a build tag footnote to a topic, the compiler includes or excludes the topic according to build information specified in the **BUILD** option and [BuildTags] section of the Help project file. For information about the **BUILD** option, the [BuildTags] section and the Help project file, see Chaper 4, "Building the Help File."

To assign a build tag to a topic:

1. Place the cursor at the beginning of the topic heading line, so that it appears before all other footnotes for that topic.
2. Insert the asterisk (*) as a footnote reference mark.

   Note that a superscript asterisk (*) appears next to the heading.
3. Type the build tag name as the footnote.

   Be sure to allow only a single space between the asterisk (*) and the build tag.

Build tags can be made up of any alphanumeric characters. The build tag is not case-sensitive. The tag may not contain spaces. You can specify multiple build tags by separating them with a semicolon, as in the following example:

```
* AppVersion1; AppVersion2; Test_Build
```

Including a build tag footnote with a topic is equivalent to setting the tag to true when compared to the value set in the project file. The compiler assumes all other build tags to be false for that topic. After setting the truth value of the build tag footnotes, the compiler evaluates the build expression in the Options section of the Help project file. Note that all build tags must be declared in the project file, regardless of whether a given conditional compilation declares the tags. If the evaluation results in a true state, the compiler includes the topic in the build. Otherwise, the compiler omits the topic.

The compiler includes in all builds topics that do not have a build tag footnote regardless of the build tag expressions defined in the Help project file. For this reason, you may want to use build tags primarily to exclude specific topics from certain builds. If the compiler finds any build tags not declared in the Help project file, it displays an error message.

By allowing conditional inclusion and exclusion of specific topics, you can create multiple builds using the same topic files. This saves

time and effort for the Help development team. It also means that you can develop Help topics that will help you maintain a higher level of consistency across your product lines.

Assigning Context Strings

Context strings identify each topic in the Help system. Each context string must be unique. A given context string may be assigned to only one topic within the Help project; it cannot be used for any other topic.

The context string provides the means for creating jumps between topics or to display look-up boxes, such as word and phrase definitions. Though not required, most topics in the Help system will have context-string identifiers, Topics without context strings may not be accessed through hypertext jumps. However, topics without context-string identifiers can be accessed through browse sequences or key-word searches, if desired. It is up to the Help writer to justify the authoring of topics that can be accessed only in these manners. For information about assigning jumps, see the section titled "Creating Cross-References Between Topics." For information about assigning browse sequences, see the section titled "Assigning Browse Sequence Numbers." For information about assigning keywords, see the section titled "Assigning Keywords."

To assign a context string to a Help topic:

1. Place the cursor to the left of the topic heading.
2. Insert the pound sign (#) as the footnote reference mark.

   Note that a superscript pound sign ($^{\#}$) appears next to the heading.
3. Type the context string as the footnote.

   Be sure to allow only a single space between the pound sign (#) and the string.

   Context strings are not case-sensitive.

Valid context strings may contain the alphabetic characters A - Z, the numeric characters 0 - 9, and the period (.) and underscore (_) characters. The following example shows the context string footnote that identifies a topic called "Opening an Existing Text File":

```
#OpeningExistingTextFile
```

Although a context string has a practical limitation of about 255 characters, there is no good reason for approaching this value. Keep the strings sensible and short so that they're easier to enter into the text files.

Assigning Titles

Title footnotes perform the following functions within the Help system:

- They appear on the Bookmark menu.
- They appear in the "Topics found " list that results from a keyword search. (Topics that do not have titles, but are accessible via keywords are listed as >>Untitled Topic<< in the Topics found list.)

Although not required, most topics have a title. You might not assign a title to topics containing low-level information that Help's search feature, look-up boxes and system messages do not access.

To assign a title to a topic:

1. Place the cursor to the left of the topic heading.
2. Insert a footnote with a dollar sign ($) as the footnote reference mark.

   Note that a superscript dollar sign ($^{\$}$) appears next to the heading.
3. Type the title as the footnote.

   Be sure to allow only a single space between the dollar sign ($) and the title.

The following is an example of a footnote that defines the title for a topic:

```
$ Help Keys
```

When adding titles, keep in mind the following restrictions:

Item	Restrictions
Characters	Titles can be up to 128 characters in length. The Help compiler truncates title strings longer than 128 characters. The help system displays titles in a list box when the user searches for a keyword or enters a bookmark.
Formatting	Title footnote entries cannot be formatted.

Assigning Keywords

Help allows the user to search for topics with the use of keywords assigned to the topics. When the user searches for a topic by keyword, Help matches the user-entered word to keywords assigned to specific topics. Help then lists matching topics by their titles in the Search dialog box. Because a keyword search is often a fast way for users to access Help topics, you'll probably want to assign key words to most topics in your Help system.

NOTE: You should specify a keyword footnote only if the topic has a title footnote since the title of the topic will appear in the search dialog when the user searches for the keyword.

To assign a keyword to a topic:

1. Place the cursor to the left of the topic heading.
2. Insert an uppercase K as the footnote reference mark.

   Note that a superscript K (^{K}) appears next to the heading.
3. Type the keyword, or keywords, as the footnote.

   Be sure to allow only a single space between the K and the first keyword.

   If you add more than one keyword, separate each with a semicolon.

The following is an example of keywords for a topic:

```
K open;opening;text file;ASCII;existing;text
only;documents;
```

Whenever the user performs a search on any of these keywords, the corresponding titles appear in a list box. More than one topic may have the same keyword.

When adding keywords, keep in mind the following restrictions:

Item	Restrictions
Characters	Keywords can include any ANSI character, including accented characters. The maximum length for keywords is 255 characters. A space imbedded in a key phrase is considered to be a character, permitting phrases to be keywords.
Phrases	Help searches for any word in the specified phrase.
Formatting	Keywords are unformatted.
Case sensitivity	Keywords are not case-sensitive.
Punctuation	Except for semicolon delimiters, you can use punctuation.

Creating Multiple Key-Word Tables

Multiple keyword tables are useful to allow a program to look up topics that are defined in alternate keyword tables. You can use an additional keyword table to allow users familiar with keywords in a different application to discover the matching keywords in your application.

Authoring additional keyword tables is a two-part process. First, the **MULTIKEY** option must be placed in the [Options] section of the project file. For information on the **MULTIKEY** option, see the section in Chapter 4 titled "Multiple Keyword Tables: The Multikey Option."

Second, the topics to be associated with the additional keyword table must be authored and labeled. Footnotes are assigned in the same manner as the normal keyword footnotes, except that the letter specified with the **MULTIKEY** option is used. With this version of the Help Compiler, the keyword footnote used is case-sensitive. Therefore, care should be taken to use the same case, usually uppercase, for your key-word footnote. Be sure to associate only one topic with a keyword. Help does not display the normal search dia-

log box for a multiple keyword search. Instead it displays the first topic found with the specified keyword. If you want the topics in your additional key-word table to appear in the normal Help key-word table, you must also specify a "K" footnote and the given key-word.

The application you are developing Help for can then display the Help topic associated with a given string in a specified keyword table. Keywords are sorted without regard to case for the keyword table. For information on the parameters passed by the application to call a topic found in alternate keyword table, see the section in Chapter 4 titled "Accessing Additional Key-Word Tables."

Assigning Browse Sequence Numbers

The Browse >> and Browse << buttons on the icon bar in the Help window let users move back and forth between related topics. The order of topics that users follow when moving from topic to topic is called a "browse sequence." A browse sequence is determined by sequence numbers, established by the Help writer.

To build browse sequences into the Help topics, the writer must:

1. Decide which topics should be grouped together and what order they should follow when viewed as a group.

   Help supports multiple, discontiguous sequence lists.

2. Code topics to implement the sequence.

NOTE In this version of Help, topics defined in browse sequences are accessed using the Browse buttons at the top of the Help window. Future versions of Help will not normally display browse buttons for the user. However, if your Help resource file includes browse sequences authored in the format described here, these future versions will support the feature by automatically displaying browse buttons for the user.

Organizing Browse Sequences

When organizing browse sequences, the writer must arrange the topics in an order that will make sense to the user. Topics can be arranged in alphabetical order within a subject, in order of difficulty, or in a sensible order that seems natural to the application. The following example illustrates browse sequences for the menu commands used in a given application. The Help writer has subjectively defined the order that makes the most sense from a procedural point of view. You may, of course, choose a different order.

```
SampleApp Commands
    File Menu - commands:005
        New Command - file_menu:005
        Open Command - file_menu:010
        Save Command - file_menu:015
        Save As Command - file_menu:020
        Print Command - file_menu:025
        Printer Setup Command - file_menu:030
        Exit Command - file_menu:035
    Edit Menu - commands:010
        Undo Command - edit_menu:025
        Cut Command - edit_menu:015
        Copy Command - edit_menu:010
        Paste Command - edit_menu:020
        Clear Command - edit_menu:005
        Select All Command - edit_menu:030
        Word Wrap Command - edit_menu:035
        Type Face Command - edit_menu:040
        Point Size Command - edit_menu:045
    Search Menu - commands:015
        Find Command - search_menu:005
        Find Next Command - search_menu:010
        Previous Command - search_menu:015
    Window Menu - commands:020
        Tile Command - window_menu:005
        Cascade Command - window_menu:010
        Arrange Icons Command - window_menu:015
        Close All Command - window_menu:020
        Document Names Command - window_menu:025
```

Each line consists of a sequence list name followed by a colon and a sequence number. The sequence list name is optional. If the sequence does not have a list name, as in the following example, the compiler places the topic in a "null" list:

```
Window Menu - 120
```

Note that the numbers used in the browse sequence example begin at 005 and advance in increments of 005. Generally, it is good practice to skip one or more numbers in a sequence so you can add new topics later if necessary. Skipped numbers are of no conseqence to the Help Compiler; only their order is significant.

Sequence numbers establish the order of topics within a browse sequence list. Sequence numbers can consist of any alphanumeric characters. During the compiling process, strings are sorted using the ASCII sorting technique, not a numeric sort.

Both the alphabetic and numeric portions of a sequence can be several characters long; however, their lengths should be consistent throughout all topic files. If you use only numbers in the strings make sure the strings are all the same length; otherwise a higher sequence number could appear before a lower one in certain cases.

For example, the number 100 is numerically higher than 99, but 100 will appear before 99 in the sort used by Help, because Help is comparing the first two digits in the strings. In order to keep the topics in their correct numeric order, you would have to make 99 a three-digit string: 099.

Coding Browse Sequences

After determining how to group and order topics, code the sequence by assigning the appropriate sequence list name and number to each topic, as follows:

1. Place the cursor to the left of the topic heading.
2. Insert the plus sign (+) as the footnote reference mark.

   Note that a superscript plus sign ($^{+}$) appears next to the heading.
3. Type the sequence number using alphanumeric characters.

For example, the following footnote defines the browse sequence number for the Edit menu topic in the previous browse sequence example:

```
+ commands:010
```

While it may be easier to list topics within the file in the same order that they appear in a browse sequence, it is not necessary. The compiler orders the sequence for you.

Creating Cross-References Between Topics

Cross-references, or "jumps," are specially-coded words or phrases that are linked to other topics. Although you indicate jump terms with strikethrough or double-underlined text in the topic file, they appear underlined in the Help window. In addition, jump terms appear in color on color systems. For example, the strikethrough text (double-underlined in Word for Windows) ~~New Command~~ appears as New Command in green text to the user.

To code a word or phrase as a jump in the topic file :

1. Place the cursor at the point in the text where you want to enter the jump term.
2. Select the strikethrough (or double-underline) feature of your editor.
3. Type the jump word or words in strikethrough mode.

4. Turn off strikethrough and select the editor's hidden text feature.
5. Type the context string assigned to the topic that is the target of the jump.

When coding jumps, keep in mind that:

- No spaces can occur between the strikethrough (or double-underlined) text and the hidden text.
- Coded spaces before or after the jump term are not permitted.
- Paragraph marks must be entered as plain text.

Defining Terms

Most topic files contain words or phrases that require further definition. To get the definition of a word or phrase, the user first selects the word and then holds down the mouse button or ENTER key, causing the definition to appear in a box within the Help window. The Help writer decides which words to define, considering the audience that will be using the application and which terms might already be familiar.

NOTE: The look-up feature need not be limited to definitions. With the capability of temporarily displaying information in a box, you might want to show a hint about a procedure, or other suitable information for the user.

Defining a term requires that you:

- Create a topic that defines the term.

  The definition topic must include a context string. See the section titled "Assigning Context Strings."

- Provide a cross-reference for the definition topic whenever the term occurs.

  You don't need to define the same word multiple times in the same topic, just its first occurrence. Also, consider the amount of colored text you are creating in the Help window.

  See the following "Coding Definitions" section.

Creating Definition Topics

You can organize definition topics any way you want. For example, you can include each definition topic in the topic file that mentions the term. Or you can organize all definitions in one topic file and provide the user with direct access to it. Helpex uses the latter

method, with all definitions residing in the TERMS.RTF file. Organizing definition topics into one file provides you with a glossary and lets you make changes easily.

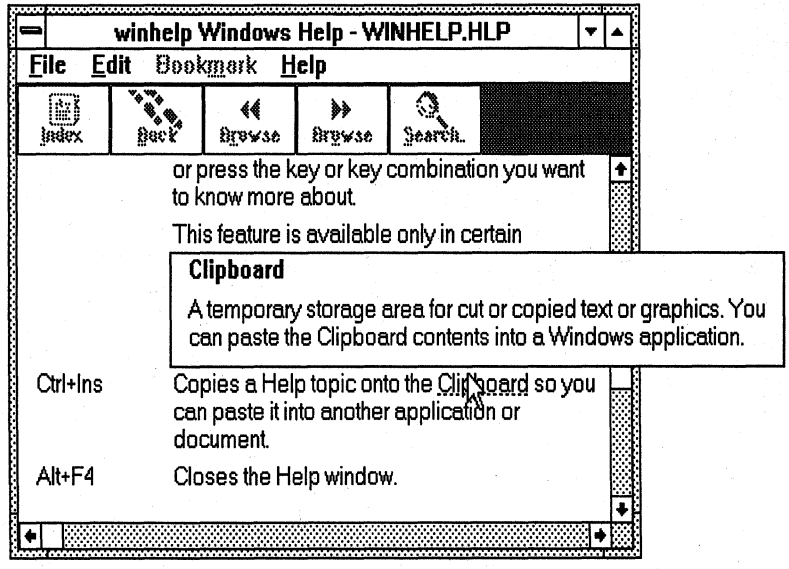
Coding Definitions

After creating definition topics, code the terms as they occur, as follows:

1. Place the insertion point where you want to place the term that requires definition.
2. Select the underline feature of your editor.
3. Type the term.
4. Turn off the underline feature, and select the editor's hidden-text feature.
5. Type the context string assigned to the topic that contains the definition of the term.

Figure 3.2 includes a definition of the term "Clipboard."

Figure 3.2
Definition as it appears in look-up box


winhelp Windows Help - WINHELP.HLP
File Edit Bookmark Help
Index
Back
Browse
Browse
Search
or press the key or key combination you want to know more about.
This feature is available only in certain
Clipboard
A temporary storage area for cut or copied text or graphics. You can paste the Clipboard contents into a Windows application.
Ctrl+Ins
Copies a Help topic onto the Clipboard so you can paste it into another application or document.
Alt+F4
Closes the Help window.

Inserting Graphic Images

Bitmapped graphic images can be placed in Help topics using either of two methods. If your word processor supports the placement of Windows 2.1 or Windows 3.0 graphics directly into a document, you can simply paste your bitmaps into each topic file. Alternatively, you can save each bitmap in a separate file and specify the file by name where you want it to appear in the Help topic file. The latter method of placing graphics is referred to as "bitmaps by reference." The following sections describe the process of placing bitmaps directly or by reference into your Help topics.

Creating and Capturing Bitmaps

You can create your bitmaps using any graphical tools, as long as the resulting images can be displayed in the Windows environment. Each graphic image can then be copied to the Windows clipboard. Once on the clipboard, a graphic can be pasted into a graphics editor such as Paint, and modified or cleaned up as needed.

Windows Help 3.0 supports color bitmaps. However, for future compatibility, you might want to limit graphics to monochrome format. If you are producing monochrome images, you might have to adjust manually the elements of your source graphic that were originally different colors to either black, white, or a pattern of black and white pixels.

When you are satisfied with the appearance of your bitmap, you can either save it as a file, to be used as a bitmap by reference, or you can copy it onto the clipboard and paste it into your word processor. If you save the graphic as a file, be sure to specify its size in your graphics editor first, so that only the area of interest is saved for display in the Help window. The tighter you crop your images, the more closely you will be able to position text next to the image. Always save (or convert and save if necessary) graphics in Windows' .BMP format.

Bitmap images should be created in the same screen mode that you intend Help to use when topics are displayed. If your Help files will be displayed in a different mode, bitmaps might not retain the same aspect ratio or information as their source images.

Placing Bitmaps Using a Graphical Word Processor

The easiest way to precisely place bitmaps into Help topics is to use a graphical word processor. Microsoft Word for Windows supports the direct importation of bitmaps from the clipboard. Simply paste the graphic image where you want it to appear in the Help topic. You can format your text so that it is positioned below or alongside the bitmap. When you save your Help topic file in RTF, the pasted-in bitmap is converted as well and will automatically be included in the Help resource file.

Placing Bitmaps by Reference

If your word processor cannot import and display bitmaps directly, you can specify the location of a bitmap that you have saved as a file. To insert a bitmap reference in the Help topic file, insert one the following statements where you want the bitmap to appear in the topic:

{**bmc** *filename*.bmp}
{**bml** *filename*.bmp}
{**bmr** *filename*.bmp}

NOTE : Do not specify a full path for *filename*. If you need to direct the compiler to a bitmap in a location other than the root directory for the build, specify the absolute path for the bitmap in the [Bitmaps] section of the project file.

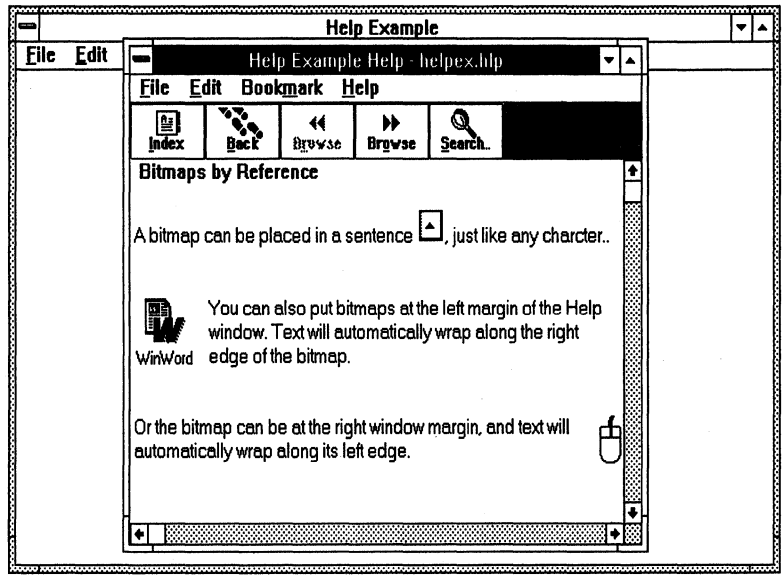
The argument **bmc** stands for "bitmap character," indicating that the bitmap referred to will be treated the same as a character placed in the topic file at the same location on a line. Text can precede or follow the bitmap on the same line, and line spacing will be determined based upon the size of the characters (including the bitmap character) on the line. Do not specify negative line spacing for a paragraph with a bitmap image, or the image may inadvertently overwrite text above it when it is displayed in Help. When you use the argument **bmc**, there is no automatic text wrapping around the graphic image. Text will follow the bitmap, positioned at the baseline.

The argument **bml** specifies that the bitmap appear at the left margin, with text wrapping automatically along the right edge of the image. The argument **bmr** specifies that the bitmap appear at the right margin, with text to its left. Bitmap filenames must be the same as those listed in the [Bitmaps] section of the Help project file. The [Bitmaps] section is described in Chapter 4, "Building the Help File."

NOTE : Multiple references to a bitmap of the same name refer to the same bitmap when the Help file is displayed. This means that bitmap references can be repeated in your Help system without markedly increasing the size of the Help resource file.

Figure 3.3 shows the placement of three bitmaps with related text in a topic as displayed in Help.

Figure 3.3
Help topic display showing bitmaps by reference


Help Example
File Edit
Help Example Help - helpex.hlp
File Edit Bookmark Help
Index
Back
Browse
Browse
Search..
Bitmaps by Reference
A bitmap can be placed in a sentence , just like any charcter..
WinWord
You can also put bitmaps at the left margin of the Help window. Text will automatically wrap along the right edge of the bitmap.
Or the bitmap can be at the right window margin, and text will automatically wrap along its left edge.

Managing Topic Files

Help topic files can be saved in the default word-processor format or in RTF. If you always save your files in RTF, and later need to make a change, the word processor may take additional time to interpret the format as it reloads the file. If you anticipate making numerous changes during Help development, you might want to minimize this delay by saving topic files in both default and RTF formats, with different file extensions to distinguish them. The compiler needs only the RTF files, and you will have faster access to the source files for changes. On a large project, this practice can save a considerable amount of development time.

Keeping Track of Files and Topics

It is important to keep track of all topic files for the following reasons:

- To ensure that no topics are left out of the build process
- To ensure that each topic has been assigned a unique context string
- To double-check browse sequencing within general and specific lists
- To show keyword and title matches
- To allow writers to see where the text for each of the topics is located
- To keep track of changes to files and the current status
- To track any other aspect of the Help development process that you think essential

At a minimum, writers must keep track of their own topic files, and must pass the filenames to the person who is responsible for creating the Help project file.

Creating a Help Tracker

While it is important that you track topic files throughout the development cycle, the tracking tool can be anything that suits your needs. You can maintain a current list of topics in an ASCII text file, in a Quattro Pro spreadsheet, or in another format.

When you or another writer creates or revises a topic, you should update the Help tracking file to reflect the change. The contents of the tracking file are not rigidly defined, but should contain entries for filename, context string, title, browse sequence, and keywords. If your application makes use of the context-sensitive feature of Help, you may want to keep track of the context-sensitive information as well. This entry is necessary only if you are assigning context numbers to topics in the Help project file. You can also include optional information, such as date created, date modified, status, and author, if you want to keep track of all aspects of the Help development process. How you organize this information is entirely up to you.

The following sample text file and worksheet illustrate how the tracker might be organized for the Help system topics. The examples show both Help menu and context-sensitive Help entries for the topic files. Typically, the same topics that the user accesses when choosing commands from the Help menus can be accessed by the context-sensitive Help feature. The topics with entries in the context ID column are used for context-sensitive help as well as for the Help menus. Notice that some topics have more than one context-sensitive help number. This enables the topic to be displayed when the user clicks on different regions of the screen.

Figure 3.4 Help tracker text file example

Ctx. String	Title	Browse Seq.	Key Words	Ctx. No.	Filename	Modified	Status
hlpidx_idx_mp	Multipad Help Index			0xFFFF	helpex.idx	5/16/89	Done
mc_cmd_mp	Multipad Commands	commands:0001	commands; menus	0x1000	helpex.cmd	5/16/89	Done
fm_cmd_mp	File Menu	commands:0004	commands; menus; files; documents	0x1001	helpex.cmd	5/16/89	Done
nc_cmd_mp	New command	commands:0008	commands; new files; new documents	0x1002	helpex.cmd	5/16/89	Done
oc_cmd_mp	Open Command	commands:0012	commands; file; open; documents; read only	0x1003	helpex.cmd	5/16/89	Text
sac_cmd_mp	Save Command	commands:0016	commands; file; save; save as; documents; files	0x1004	helpex.cmd	5/16/89	Done
sasc_cmd_mp	Save As Command	commands:0020	commands; file; save as; save; documents; files	0x1005	helpex.cmd	5/16/89	Done
ptc_cmd_mp	Print Command	commands:0024	commands; file; print; documents; files	0x1006	helpex.cmd	5/16/89	Done
psc_cmd_mp	Print Setup Command	commands:0028	commands; file; printer setup; print	0x1007	helpex.cmd	5/16/89	Debug
ec_cmd_mp	Exit Command	commands:0032	commands; file; exit; exiting; close; closing; quit;quitting	0x1008	helpex.cmd	5/16/89	Done
em_cmd_mp	Edit Menu	commands:0036	commands; menus; editing; documents	0x1009	helpex.cmd	5/16/89	Done
uc_cmd_mp	Undo Command	commands:0040	commands; edit; editing; undo; typing	0x100A	helpex.cmd	5/16/89	Done
ctc_cmd_mp	Cut Command	commands:0044	commands; edit; editing; cut; cutting; text; Clipboard	0x100B	helpex.cmd	5/16/89	Done
cyc_cmd_mp	Copy Command	commands:0048	commands; edit editing; copy; copying; text; Clipboard	0x100C	helpex.cmd	5/16/89	Done
pec_cmd_mp	Past Command	commands:0052	commands; edit editing; paste; insert; inserting; text; Clipboard	0x100D	helpex.cmd	5/16/89	Done
crc_cmd_mp	Clear Command	commands:0056	commands; edit; editing; clear; text	0x100E	helpex.cmd	5/16/89	Done
salc_cmd_mp	Select All Command	commands:0060	commands; edit editing; select all; select; selecting; text	0x100F	helpex.cmd	5/16/89	Done
wwc_cmd_mp	Word Wrap Command	commands:0064	commands; edit; editing; word wrap; wrapping; text; format	0x1010	helpex.cmd	5/16/89	Done
tfc_cmd_mp	Type Face Command	commands:0068	commands; edit editing; type face; font; text; format	0x1011	helpex.cmd	5/16/89	Test
ptsc_cmd_mp	Point Size Command	commands:0072	commands; edit; editing; point size; text; format	0x1012	helpex.cmd	5/16/89	Test

Figure 3.5 Help tracker worksheet example

	A	B	C	D	E	F	G	H
1	Ctx. String	Title	Browse Seq.	Key Words	Ctx. No.	Filename	Modified	Status
2	hlpidx_idx_mp	Multipad Help Index			0xFFFF	helpex.idx	5/16/89	Done
3	mc_cmd_mp	Multipad Commands	commands:0001	commands; menus	0x1000	helpex.cmd	5/16/89	Done
4	fm_cmd_mp	File Menu	commands:0004	commands; menus; files;	0x1001	helpex.cmd	5/16/89	Done
5				documents				
6	nc_cmd_mp	New command	commands:0008	commands; new files;	0x1002	helpex.cmd	5/16/89	Done
7				new documents				
8	oc_cmd_mp	Open Command	commands:0012	commands; file; open;	0x1003	helpex.cmd	5/16/89	Text
9				documents; read only				
10	sac_cmd_mp	Save Command	commands:0016	commands; file; save; save as;	0x1004	helpex.cmd	5/16/89	Done
11				documents; files				
12	sasc_cmd_mp	Save As Command	commands:0020	commands; file; save as; save;	0x1005	helpex.cmd	5/16/89	Done
13				documents; files				
14	ptc_cmd_mp	Print Command	commands:0024	commands; file; print;	0x1006	helpex.cmd	5/16/89	Done
15				documents; files				
16	psc_cmd_mp	Print Setup Command	commands:0028	commands; file; printer setup;	0x1007	helpex.cmd	5/16/89	Debug
17				print				
18	ec_cmd_mp	Exit Command	commands:0032	commands; file; exit; exiting;	0x1008	helpex.cmd	5/16/89	Done
19				close; closing; quit;quitting				
20	em_cmd_mp	Edit Menu	commands:0036	commands; menus; editing;	0x1009	helpex.cmd	5/16/89	Done
21				documents				
22	uc_cmd_mp	Undo Command	commands:0040	commands; edit; editing; undo;	0x100A	helpex.cmd	5/16/89	Done
23				typing				
24	ctc_cmd_mp	Cut Command	commands:0044	commands; edit; editing; cut;	0x100B	helpex.cmd	5/16/89	Done
25				cutting; text; Clipboard				
26	cyc_cmd_mp	Copy Command	commands:0048	commands; edit editing; copy;	0x100C	helpex.cmd	5/16/89	Done
27				copying; text; Clipboard				
28	pec_cmd_mp	Past Command	commands:0052	commands; edit editing; paste;	0x100D	helpex.cmd	5/16/89	Done
29				insert; inserting; text; Clipboard				
30	crc_cmd_mp	Clear Command	commands:0056	commands; edit; editing; clear;	0x100E	helpex.cmd	5/16/89	Done
31				text				
32	salc_cmd_mp	Select All Command	commands:0060	commands; edit editing; select all;	0x100F	helpex.cmd	5/16/89	Done
33				select; selecting; text				
34	wwc_cmd_mp	Word Wrap Command	commands:0064	commands; edit; editing; word wrap;	0x1010	helpex.cmd	5/16/89	Done
35				wrapping; text; format				
36	tfc_cmd_mp	Type Face Command	commands:0068	commands; edit editing; type face;	0x1011	helpex.cmd	5/16/89	Test
37				font; text; format				
38	ptsc_cmd_mp	Point Size Command	commands:0072	commands; edit; editing; point size;	0x1012	helpex.cmd	5/16/89	Test
39				text; format				

Of course, you are free to keep track of the topic files you produce in a manner different from either of these two examples.

Summary

This chapter described how to write, code, and keep track of Help topic files. For information about building a Help file, see Chapter 4, "Building the Help File."

Building the Help File

After the topic files for your Help system have been written, you are ready to create a Help project file and run a build to test the Help file. The Help project file contains all information the compiler needs to convert help topic files into a binary Help resource file.

You use the Help project file to tell the Help Compiler which topic files to include in the build process. Information in the Help project file also enables the compiler to map specific topics to context numbers (for the context-sensitive portion of Help).

After you have compiled your Help file, the development team programs the application so the user can access it.

This chapter describes the following:

- Creating a Help project file
- Compiling the Help file
- Programming the application to access Help

While most of the examples in this book are in C, you can also do the same tasks in Turbo Pascal. Code examples for both languages are included on your Borland product disks.

Creating the Help Project File

You use the Help project file to control how the Help Compiler builds your topic files. The Help project file can contain up to six sections that perform the following functions:

Section	Function
[Files]	Specifies topic files to be included in the build. This section is mandatory.
[Options]	Specifies the level of error reporting, topics to be included in the build, the directory in which to find the files, and the location of your Help index. This section is optional.
[BuildTags]	Specifies valid build tags. This section is optional.
[Alias]	Assigns one or more context strings to the same topic. This section is optional.
[Map]	Associates context strings with context numbers. This section is optional.
[Bitmaps]	Specifies bitmap files to be included in the build. This section is optional.

You can use any ASCII text editor to create your Help project file. The extension of a Help project file is .HPJ. If you do not use the extension .HPJ on the **HC** command line, the compiler looks for a project file with this extension before loading the file. The .HLP output file will have the same name as the .HPJ file.

The order of the various sections within the Help project file is arbitrary, with one exception: under all circumstances an [Alias] section must precede the [Map] section (if an [Alias] section is used).

Section names are placed within square brackets using the following syntax:

[*section-name*]

You can use a semicolon to indicate a comment in the project file. The compiler ignores all text from the semicolon to the end of the line on which it occurs.

Specifying Topic Files: The Files Section

Use the [Files] section of the Help project file to list all topic files that the Help Compiler should process to produce a Help resource file. A Help project file must have a [Files] section.

The following sample shows the format of the [Files] section:

```
[FILES]
HELPEX.RTF ;Main topics for HelpEx application
TERMS.RTF  ;Lookup terms for HelpEx application
```

Using the path defined in the **ROOT** option, the Help Compiler finds and processes all files listed in this section of the Help project file. If the file is not on the defined path and cannot be found, the compiler generates an error. For more information about the **ROOT** option, see the section titled "Specifying the Root Directory: The Root Option."

You can include files in the build process using the C **#include** directive command. The **#include** directive uses the following syntax:

#include *<filename>*

You must include the angle brackets around the filename. The pound sign (#) must be the first character in the line. The filename must specify a complete path, either the path defined by the **ROOT** option or an absolute directory path to the file.

You may find it easier to create a text file that lists all files in the Help project and include that file in the build, as in this example:

```
[FILES]
#include <hlpfiles.inc>
```

Specifying Build Tags: The BuildTags Section

If you code build tags in your topic files, use the [BuildTags] section of the Help project file to define all the valid build tags for a particular Help project. The [BuildTags] section is optional.

The following example shows the format of the [BuildTags] section in a sample Help project file:

```
[BUILDTAGS]
WINENV     ;topics to include in Windows build
DEBUGBUILD ;topics to include in debugging build
TESTBUILD  ;topics to include in a mini-build for testing
```

The [BuildTags] section can include up to 30 build tags. The build tags are not case-sensitive and may not contain space characters. Only one build tag is allowed per line in this section. The compiler will generate an error message if anything other than a comment is listed after a build tag in the [BuildTags] section.

For information about coding build tags in topic files, see the section in Chapter 3 titled "Assigning Build Tags."

Specifying Options: The Options Section

Use the [Options] section of the Help project file to specify the following options:

Option	Meaning
BUILD	Determines what topics the compiler includes in the build.
COMPRESS	Specifies compression of the Help resource file.
FORCEFONT	Specifies the creation of a Help resource file using only one font.
INDEX	Specifies the context string of the Help index.
MAPFONT-SIZE	Determines the mapping of specified font sizes to different sizes.
MULTIKEY	Specifies alternate key-word mapping for topics.
ROOT	Designates the directory to be used for the Help build.
TITLE	Specifies the title shown for the Help system.
WARNING	Indicates the kind of error message the compiler reports.

These options can appear in any order within the [Options] section. The [Options] section is not required.

Detailed explanations of the available options follow.

Specifying Error Reporting: The Warning Option

Use the **WARNING** option to specify the amount of debugging information that the compiler reports. The **WARNING** option has the following syntax:

WARNING = *level*

You can set the **WARNING** option to any of the following levels:

Level	Information Reported
1	Only the most severe warnings.
2	An intermediate level of warnings.
3	All warnings. This is the default level if no **WARNING** option is specified.

The following example specifies an intermediate level of error reporting:

```
[OPTIONS]
WARNING=2
```

The compiler reports errors to the standard output file, typically the screen. You may want to redirect the errors to a disk file so that you can browse it when you are debugging the Help system. The following example shows the redirection of compiler screen output to a file.

```
HC HELPEX > errors.out
```

HINT: Use the DOS CONTROL+PRINT SCREEN accelerator key before you begin your compilation to echo errors which appear on the screen to your printer. Type CONTROL+PRINT SCREEN again to stop sending information to the printer.

Specifying Build Topics: The Build Option

If you have included build tags in your topic files, use the **BUILD** option to specify which topics to conditionally include in the build. If your topic files have no build tags, omit the **BUILD** option from the [Options] section.

NOTE: All build tags must be listed in the [BuildTags] section of the project file, regardless whether or not a given conditional compilation declares the tags.

See Chapter 3, "Creating the Help Topic Files," for information on assigning build tags to topics in the Help topic files.

The **BUILD** option line uses the following syntax:

BUILD = *expression*

BUILD expressions cannot exceed 255 characters in length, and must be entered on only one line. **BUILD** expressions use Boolean logic to specify which topics within the specified Help topic files the compiler will include in the build. The compiler evaluates all

build expressions from left to right. The tokens of the language (listed in order of precedence from highest to lowest) are:

Token	Description
<tag >	Build tag
( )	Parentheses
~	NOT operator
&	AND operator
|	OR operator

For example, if you coded build tags called WINENV, APP1, and TEST_BUILD in your topic files, you could include one of the following build expressions in the [Options] section:

Build Expression	Topics Built
BUILD = WINENV	Only topics that have the WINENV tag
BUILD = WINENV & APP1	Topics that have both the WINENV and APP1 tags
BUILD = WINENV | APP1	Topics that have either the WINENV tag or the APP1 tag
BUILD = (WINENV | APP1) & TESTBUILD	Topics that have either the WINENV or the APP1 tags and that also have the TESTBUILD tag
BUILD =~ APP1	Topics that do not have an APP1 tag

Specifying the Root Directory: The Root Option

Use the **ROOT** option to designate the root directory of the Help project. The compiler searches for files in the specified root directory.

The **ROOT** option uses the following syntax:

ROOT = *pathname*

For example, the following root option specifies that the root directory is \BUILD\TEST on drive D:

```
[OPTIONS]
ROOT=D:\BUILD\TEST
```

The **ROOT** option allows you to refer to all relative paths off the root directory of the Help project. For example, the following entry in the [Files] section refers to a relative path off the root directory:

```
TOPICS\FILE.RTF
```

To refer to a file in a fixed location, independent of the project root, you must specify a fully qualified or "absolute" path, including a drive letter, if necessary, as in the following line:

```
D:\HELPTEST\TESTFILE.RTF
```

If you do not include the **ROOT** option in your Help project file, all paths are relative to the current DOS directory.

Specifying the Index: The Index Option

Use the **INDEX** option to identify the context string of the Help index. Because the Index button gives the user access to the index from anywhere in the Help system, you will probably not want to author terms to jump to the index. Users access this general index either from the Help menu of the application or by choosing the Index button from the Help window.

Assigning a context string to the index topic in the [Options] section lets the compiler know the location of the main index of Help topics for the application's Help file. If you do not include the **INDEX** option in the [Options] section, the compiler assumes that the first topic it encounters is the index.

The **INDEX** option uses the following syntax:

INDEX = *context-string*

The context string specified must match the context string you assigned to the Help index topic. In the following example, the writer informs the compiler that the context string of the Help index is "main_index":

```
[OPTIONS]
INDEX=main_index
```

For information about assigning context strings, see the section in Chapter 3 titled "Assigning Context Strings."

Assigning a Title to the Help System: The Title Option

You can assign a title to your Help system with the **TITLE** option. The title appears in the title bar of the Help window with the word "Help" automatically appended, followed by the DOS filename of the Help resource file.

The **TITLE** option uses the following syntax:

TITLE = *Help-system-title-name*

Titles are limited to 32 characters in length. If you do not specify a title using the **TITLE** option, only the word Help followed by the Help system filename will be displayed in the title bar. Because the compiler always inserts the word Help, you should keep in mind not to duplicate it in your title.

Converting Fonts: The Forcefont Option

You can use the **FORCEFONT** option to create a Help resource file that is made up of only one typeface or font. This is useful if you must compile a Help system using topic files that include fonts not supported by your users' systems.

The **FORCEFONT** option uses the following syntax:

FORCEFONT = *fontname*

The *fontname* parameter is any Windows system font. Windows ships with the following fonts and sizes:

- Courier 10,12,15
- Helv 8,10,12,14,18,24
- Modern
- Roman
- Script
- Symbol 8,10,12,14,18,24
- Tms Rmn 8,10,12,14,18,24

Font names must be spelled the same as they are in the Fonts dialog box in Control Panel. Font names do not exceed 20 characters in length. If you designate a font that is not recognized by the compiler, an error message is generated and the compilation continues using the default Helvetica (Helv) font.

NOTE: The *fontname* used in the **FORCEFONT** option cannot contain spaces. Therefore, Tms Rmn font cannot be used with **FORCEFONT**.

Changing Font Sizes : The Mapfontsize Option

The font sizes specified in your topic files can be mapped to different sizes using the **MAPFONTSIZE** option. In this manner, you can create and edit text in a size chosen for easier viewing in the topic files and have them sized by the compiler for the actual Help display. This may be useful if there is a large size difference between your authoring monitor and your intended display monitor.

The **MAPFONTSIZE** option uses the following syntax:

MAPFONTSIZE = *m*[–*n*]:*p*

The *m* parameter is the size of the source font, and the *p* parameter is the size of the desired font for the Help resource file. All fonts in the topic files that are size *m* are changed to size *p*. The optional parameter *n* allows you to specify a font range to be mapped. All fonts in the topic files falling between *m* and *n*, inclusive, are changed to size *p*. The following examples illustrate the use of the **MAPFONTSIZE** option:

```
MAPFONTSIZE=12-24:16;make fonts from 12 to 24 come out 16.
MAPFONTSIZE=8:12  ;make all size 8 fonts come out size 12.
```

Note that you can map only one font size or range with each **MAPFONTSIZE** statement used in the Options section. If you use more than one **MAPFONTSIZE** statement, the source font size or range specified in subsequent statements cannot overlap previous mappings. For instance, the following mappings would generate an error when the compiler encountered the second statement:

```
MAPFONTSIZE=12-24:16 MAPFONTSIZE=14:20
```

Because the second mapping shown in the first example contains a size already mapped in the preceding statement, the compiler will ignore the line. There is a maximum of five font ranges that can be specified in the project file.

Multiple Keyword Tables: The Multikey Option

The **MULTIKEY** option specifies a character to be used for an additional keyword table.

The **MULTIKEY** option uses the following syntax:

MULTIKEY = *footnote-character*

The *footnote-character* parameter is the case-sensitive letter to be used for the keyword footnote. The following example illustrates the enabling of the letter L for a keyword–table footnote:

```
MULTIKEY=L
```

NOTE: You must be sure to limit your key word–table footnotes to one case, usually uppercase. In the previous example, topics with the footnote L would have their key words incorporated into the additional key word table, whereas those assigned the letter l would not.

You may use any alphanumeric character for a keyword table except "K" and "k," which are reserved for Help's normal keyword table. There is an absolute limit of five keyword tables, including the normal table. However, depending upon system configuration and the structure of your Help system, a practical limit of only two or three may actually be the case. If the compiler cannot create an additional table, the excess table is ignored in the build.

Compressing the File: The Compress Option

You can use the **COMPRESS** option to reduce the size of the Help resource file created by the compiler. The amount of file compression realized will vary according to the number, size and complexity of topics that are compiled. In general, the larger the Help files, the more they can be compressed.

The **COMPRESS** option uses the following syntax:

```
COMPRESS = TRUE | FALSE
```

Because the Help application can load compressed files quickly, there is a clear advantage in creating and shipping compressed Help files with your application. Compiling with compression turned on, however, may increase the compile time, because of the additional time required to assemble and sort a key-phrase table. Thus, you may want to compile without compression in the early stages of a project.

The **COMPRESS** option causes the compiler to compress the system by combining repeated phrases found within the source file(s). The compiler creates a phrase-table file with the .PH extension if it does not find one already in existence. If the compiler finds a file with the .PH extension, it will use the file for the current compilation. This is in order to speed compression when not a lot of text has changed since the last compilation.

Deleting the key-phrase file before each compilation will prevent the compiler from using the previous file. Maximum compression will result only by forcing the compiler to create a new phrase table.

Specifying Alternate Context Strings: The Alias Section

Use the [Alias] section to assign one or more context strings to the same topic alias. Because context strings must be unique for each topic and cannot be used for any other topic in the Help project, the [Alias] section provides a way to delete or combine Help topics without recoding your files. The [Alias] section is optional.

For example, if you create a topic that replaces the information in three other topics, and you delete the three, you will have to search through your files for invalid cross-references to the deleted topics. You can avoid this problem by using the [Alias] section to assign the name of the new topic to the deleted topics. The [Alias] section can also be used when your application program has multiple context identifiers for which you have only one topic. This can be the case with context-sensitive Help.

Each expression in the [Alias] section has the following format:

context_string=alias

In the alias expression the *alias* parameter is the alternate string, or alias name, and the *context_string* parameter is the context string identifying a particular topic. An alias string has the same format and follows the same conventions as the topic context string. That is, it is not case-sensitive and may contain the alphabetic characters A - Z, numeric characters 0 - 9, and the period and underscore characters.

The following example illustrates an [Alias] section:

```
[ALIAS]
sm_key=key_shrtcuts
cc_key=key_shrtcuts
st_key=key_shrtcuts;combined into keyboard shortcuts
topic
clskey=us_dlog_bxs
maakey=us_dlog_bxs;covered in using dialog boxes topic
chk_key=dlogprts
drp_key=dlogprts
lst_key=dlogprts
opt_key=dlogprts
tbx_key=dlogprts  ;combined into parts of dialog box topic
frmtxt=edittxt
```

```
wrptxt=edittxt
seltxt=edittxt    ;covered in editing text topic
```

NOTE: You can use alias names in the [Map] section of the Help project file. If you do, however, the [Alias] section must precede the [Map] section.

Mapping Context-Sensitive Topics: The Map Section

If your Help system supports context-sensitive Help, use the [Map] section to associate either context strings or aliases to context numbers. The context number corresponds to a value the parent application passes to the Help application in order to display a particular topic. This section is optional.

When writing the [Map] section, you can do the following:

- Use either decimal or hexadecimal numbers formatted in standard C notation to specify context numbers.
- Assign no more than one context number to a context string or alias.

  Assigning the same number to more than one context string will generate a compiler error.
- Separate context numbers and context strings by an arbitrary amount of white space using either space characters or tabs.

You can use the C **#include** directive to include other files in the mapping. In addition, the Map section supports an extended format that lets you include C files with the .H extension directly. Entries using this format must begin with the **#define** directive and may contain comments in C format, as in this example:

```
#define    context_string context_number    /* comment */
```

The following example illustrates several formats you can use in the [Map] section:

```
[MAP]
❶
Edit_Window         0x0001
Control_Menu        0x0002
Maximize_Icon       0x0003
Minimize_Icon       0x0004
Split_Bar           0x0005
Scroll_Bar          0x0006
Title_Bar           0x0007
```

```
Window_Border     0x0008
❷
dcmb_scr          30; Document Control-menu Icon
dmxi_scr          31; Document Maximize Icon
dmni_scr          32; Document Minimize Icon
dri_scr           33; Document Restore Icon
dtb_scr           34; Document Title Bar

❸
#define vscroll   0x010A/* Vertical Scroll Bar */
#define hscroll   0x010E/* Horizontal Scroll Bar */
#define scrollthm 0x0111/* Scroll Thumb */
#define upscroll  0x0112/* Up Scroll Arrow */
#define dnscroll  0x0113/* Down Scroll Arrow */

❹
#include <sample.h>
```

In the example:

❶ These eight entries give hexadecimal equivalents for the context numbers.

❷ These five entries show decimal context numbers.

❸ These five entries show how you might include topics defined in a C include file.

❹ This entry shows a C **#include** directive for some generic topics.

If context numbers use the **#define** directive, and the file containing the **#define** statements is included in both the application code and the Help file, then updates made to the context numbers by the application programmers will automatically be reflected in the next Help build.

You can define the context strings listed in the [Map] section either in a Help topic or in the [Alias] section. The compiler generates a warning message if a context string appearing in the [Map] section is not defined in any of the topic files or in the [Alias] section.

NOTE: If you use an alias name, the [Alias] section must precede the [Map] section in the Help project file.

For more information about context-sensitive Help, see the section in Chapter 2 titled "Displaying Context-Sensitive Help Topics."

Including Bitmaps by Reference: The Bitmaps Section

If your Help system uses bitmaps by reference, the filenames of each of the bitmaps must be listed in the [Bitmaps] section of the project file. The following example illustrates the format of the [Bitmaps] section.

```
[BITMAPS]
DUMP01.BMP
DUMP02.BMP
DUMP03.BMP
c:\PROJECT\HELP\BITMAPS\DUMP04.BMP
```

NOTE: The [Bitmaps] section uses the same rules as the [Files] section for locating bitmap files.

Compiling Help Files

After you have created a Help project file, you are ready to build a Help file using the Help Compiler. The compiler generates the binary Help resource file from the topic files listed in the Help project file. When the build process is complete, your application can access the Help resource file that results.

Before initiating a build operation to create the Help file, consider the locations of the following files:

- The Help Compiler, HC.EXE. The compiler must be in a directory from which it can be executed. This could be the current working directory, on the path set with the PATH environment variable, or a directory specified by a full pathname, as follows:

  ```
  C:\BIN\HC HELPEX.HPJ
  ```

- The Help project file, *filename*.HPJ. The project file can be located either in the current directory or specified by a path, as follows:

  ```
  C:\BIN\HC D:\MYPROJ\HELPEX.HPJ
  ```

- The topic files named in the Help project file, saved as RTF. The topic files may be located in the current working directory, a subdirectory of the current working directory specified in the [Files] section, or the location specified in the Root option.

- Files included with the **#include** directive in the Help project file. Since the **#include** directive can take pathnames, then any number of places will work for these files.
- All bitmap files listed by reference in the topic files.

You must also place any files named in an **#include** directive in the path of the project root directory or specify their path using the **ROOT** option. The compiler searches only the directories specified in the Help project file. For information about the **ROOT** option, see the section titled "Specifying the Root Directory: The Root Option."

NOTE: If you use a RAM drive for temporary files (set with the DOS environment variable TMP), it must be large enough to hold the compiled Help resource file. If your Help file is larger than the size of the available RAM drive, the compiler will generate an error message and the compilation will be aborted.

Using the Help Compiler

To run the Help Compiler, use the **HC** command. There are no options for **HC**. All options are specified in the Help project file.

The **HC** command uses the following syntax:

HC *filename*.HPJ

As the compiler program runs, it displays sequential periods on the screen, indicating its progress in the process. Error messages are displayed when each error condition is encountered. When the Help Compiler has finished compiling, it writes a Help resource file with an .HLP extension in the current directory and returns to the DOS prompt. The Help resource file that results from the build has the same name as does the Help project file.

Compiler errors and status messages can be redirected to a file using standard DOS redirection syntax. This is useful for a lengthy build where you may not be monitoring the entire process. The redirected file is saved as a text file that can be viewed with any ASCII editor.

Programming the Application to Access Help

The application development team must program the application so that the user can access both the Windows Help application and your Help file. The Help application is a stand-alone Windows application, and your application can ask Windows to run the Help application and specify the topic that Help is to show the user. To the user, Help appears to be part of your application, but it acts like any other Windows application.

Calling WinHelp from an Application

An application makes a Help system available to the user by calling the **WinHelp** function.

The **WinHelp** function uses the following syntax:

BOOL WinHelp (*hWnd, lpHelpFile, wCommand, dwData*)

Reminder: The C and Pascal samples are provided on disk.

The *hWnd* parameter identifies the window requesting Help. The Windows Help application uses this identifier to keep track of which applications have requested Help.

The *lpHelpFile* parameter specifies the name (with optional directory path) of the Help file containing the desired topic.

The *wCommand* parameter specifies either the type of search that the Windows Help application is to use to locate the specified topic, or that the application no longer requires Help. It may be set to any one of the following values:

Value	Meaning
HELP_CONTEXT	Displays Help for a particular topic identified by a context number.
HELP_HELPONHELP	Displays the Using Help index topic.
HELP_INDEX	Displays the main Help index topic.
HELP_KEY	Displays Help for a topic identified by a key word.
HELP_MULTIKEY	Displays Help for a topic identified by a key word in an alternate keyword table.

Value	Meaning
HELP_QUIT	Informs the Help application that Help is no longer needed. If no other applications have asked for Help, Windows closes the Help application.
HELP_SETINDEX	Displays a designated Help index topic.

The *dwData* parameter specifies the topic for which the application is requesting Help. The format of *dwData* depends upon the value of *wCommand* passed when your application calls **WinHelp**. The following list describes the format of *dwData* for each value of *wCommand*.

wCommand Value	dwData Format
HELP_CONTEXT	An unsigned long integer containing the context number for the topic. Instead of using HELP_INDEX, HELP_CONTEXT can use the value -1.
HELP_HELPONHELP	Ignored.
HELP_INDEX	Ignored.
HELP_KEY	A long pointer to a string which contains a key word for the desired topic.
HELP_MULTIKEY	A long pointer to the **MULTIKEYHELP** structure, as defined in WINDOWS.H. This structure specifies the table footnote character and the key word.
HELP_QUIT	Ignored.
HELP_SETINDEX	An unsigned long integer containing the context number for the topic.

Because it can specify either a context number or a key word, **WinHelp** supports both context-sensitive and topical searches of the Help file.

NOTE: To ensure that the correct index remains set, the application should call **WinHelp** with *wCommand* set to HELP_SETINDEX (with *dwData* specifying the corresponding context identifier) following each

call to **WinHelp** with a command set to HELP_CONTEXT. HELP_INDEX should never be used with HELP_SETINDEX.

Getting Context-Sensitive Help

Context-sensitive Help should be made available when a user wants to learn about the purpose of a particular window or control. For example, the user might pull down the File menu, select the Open command (by using the DIRECTION keys), and then press F1 to get Help for the command.

Implementing certain types of context-sensitive help requires advanced programming techniques. The Helpex sample application illustrates the use of two techniques. These techniques are described in the following sections.

Shift+F1 Support

To implement a SHIFT+F1 mode, Helpex responds to the SHIFT+F1 accelerator key by calling **Set-Cursor** to change the shape of the cursor to an arrow pointer supplemented by a question mark.

```
case WM_KEYDOWN:
     if (wParam == VK_F1) {

         /* If Shift-F1, turn help mode on and set help
            cursor */

         if (GetKeyState(VK_SHIFT)) {
             bHelp = TRUE;
             SetCursor(hHelpCursor);
             return (DefWindowProc(hWnd, message,
                     wParam, lParam));
         }

         /* If F1 without shift, then call up help main
            index topic */
         else {
             WinHelp(hWnd,szHelpFileName,HELP_INDEX,0L);
         }
     }

     else if (wParam == VK_ESCAPE && bHelp) {

         /* Escape during help mode: turn help mode off */
         bHelp = FALSE;
         SetCursor((HCURSOR)GetClassWord(hWnd,GCW_HCURSOR));
     }

     break;
```

Here is how you would do this in Turbo Pascal:

```
WM_KEYDOWN:
  if WParam=VK_F1 then
  { If Shift-F1, turn help mode on and set help cursor }
    if GetKeyState(VK_Shift)  0 then
    begin
      Help:=true;
      SetCursor(HelpCursor);
      MainWndProc:=DefWindowProc(Wnd, Message, WParam,
                                 LParam);
    end
  { If F1 without shift, call up help main index topic }
    else
      WinHelp(Wnd, TempFile, Help_Index, 0)
  { Escape during help mode: turn help mode off }
  else if (WParam=VK_Escape) and Help then
  begin
    Help:=false;
    SetCursor(hCursor(GetClassWord(Wnd, GCW_HCursor)));
  end;
```

As long as the user is in Help mode (that is, until he clicks the mouse or presses *Escape*), Helpex responds to WM_SETCURSOR messages by resetting the cursor to the arrow and question mark combination.

```
case WM_SETCURSOR:
    /* In help mode it is necessary to reset the cursor
       in response to every WM_SETCURSOR message.
       Otherwise, by default, Windows will reset the
       cursor to that of the window class. */

    if (bHelp) {
        SetCursor(hHelpCursor);
        break;
    }
    return (DefWindowProc(hWnd, message, wParam, lParam));
    break;

case WM_INITMENU:
    if (bHelp) {
        SetCursor(hHelpCursor);
    }
    return (TRUE);
```

Here is how you would do this in Turbo Pascal:

```
WM_SETCURSOR:

   { In help mode it is necessary to reset the cursor
     in response to every WM_SETCURSOR message.  Otherwise,
     by default, Windows will reset the cursor to that
     of the window class.  }
```

```
    If Help then
      SetCursor(HelpCursor)
    else begin
      MainWndProc:=DefWindowProc(Wnd,Message, WParam,
                                 LParam);
      exit;
    end;
```

When the user is in SHIFT+F1 Help mode and clicks the mouse button, Helpex will receive a WM_NCLBUTTONDOWN message if the click is in a nonclient area of the application window. By examining the *wParam* value of this message, the program can determine which context ID to pass to **WinHelp**.

```
  case WM_NCLBUTTONDOWN:
      /* If we are in help mode (Shift+F1) then display
context-sensitive help for nonclient area. */

      if (bHelp) {
          dwHelpContextId =
               (wParam == HTCAPTION)?(DWORD)HELPID_TITLE_BAR:
               (wParam == HTSIZE)? (DWORD)HELPID_SIZE_BOX:
               (wParam == HTREDUCE)?
(DWORD)HELPID_MINIMIZE_ICON:
               (wParam == HTZOOM)? (DWORD)HELPID_MAXIMIZE_ICON:
               (wParam == HTSYSMENU)?(DWORD)HELPID_SYSTEM_MENU:
               (wParam == HTBOTTOM)?
(DWORD)HELPID_SIZING_BORDER:
               (wParam == HTBOTTOMLEFT)?
(DWORD)HELPID_SIZING_BORDER:
               (wParam ==
HTBOTTOMRIGHT)?(DWORD)HELPID_SIZING_BORDER:
               (wParam == HTTOP)?(DWORD)HELPID_SIZING_BORDER:
               (wParam == HTLEFT)?(DWORD)HELPID_SIZING_BORDER:
               (wParam == HTRIGHT)?(DWORD)HELPID_SIZING_BORDER:
               (wParam ==
HTTOPLEFT)?(DWORD)HELPID_SIZING_BORDER:
               (wParam == HTTOPRIGHT)?
(DWORD)HELPID_SIZING_BORDER:
               (DWORD)0L;

          if (!((BOOL)dwHelpContextId))
              return (DefWindowProc(hWnd, message, wParam,
                      lParam));
          bHelp = FALSE;

WinHelp(hWnd,szHelpFileName,HELP_CONTEXT,dwHelpContextId);
          break;
      }

      return (DefWindowProc(hWnd, message, wParam, lParam));
```

Here is how you would do this in Turbo Pascal:

```
WM_NCLBUTTONDOWN:

   { If we are in help mode (Shift+F1) then display
     context-sensitive help for non-client area.  }

   if Help then
   begin
     case WParam of
       HTCAPTION: HelpContextId:=HelpIdTitleBar;
       HTREDUCE: HelpContextId:=HelpIdMinimizeIcon;
       HTZOOM: HelpContextId:=HelpIdMaximizeIcon;
       HTSYSMENU: HelpContextId:=HelpIdSystemMenu;
       HtBottom: HelpContextId:=HelpIdSizingBorder;
       HtBottomLeft: HelpContextId:=HelpIdSizingBorder;
       HtBottomRight: HelpContextId:=HelpIdSizingBorder;
       HtTop: HelpContextId:=HelpIdSizingBorder;
       HtLeft: HelpContextId:=HelpIdSizingBorder;
       HtRight: HelpContextId:=HelpIdSizingBorder;
       HtTopLeft: HelpContextId:=HelpIdSizingBorder;
       HtTopRight: HelpContextId:=HelpIdSizingBorder;
     else
       HelpContextId:=0;
     end;
     if HelpContextId=0 then
     begin
       MainWndProc:=DefWindowProc(Wnd, Message, WParam,
                                  LParam);
       exit;
     end;
     Help:=false;
     WinHelp(Wnd, TempFile, Help_Context, HelpContextId);
   end
   else begin
     MainWndProc:=DefWindowProc(Wnd, Message, WParam,
                                LParam);
     exit;
   end;
```

F1 Support

Context-sensitive F1 support for menus is relatively easy to implement in your application. If a menu is open and the user presses F1 while one of the menu items is highlighted, Windows sends a WM_ENTERIDLE message to the application to indicate that the system is going back into an idle state after having determined that F1 was not a valid key stroke for choosing a menu item. You can take advantage of this idle state by looking at the keyboard state at the time of the WM_ENTERIDLE message.

If the F1 key is down, then you can simulate the user's pressing the ENTER key by posting a WM_KEYDOWN message using VK_RETURN. You don't really want your application to execute the menu command. What you do is set a flag (bHelp=TRUE) so that when

you get the WM_COMMAND message for the menu item, you don't execute the command. Instead, the topic for the menu item is displayed by Windows Help.

The following code samples illustrate F1 sensing for menu items.

```
  case WM_ENTERIDLE:
      if ((wParam == MSGF_MENU) && (GetKeyState(VK_F1) &
         0x8000)) {
          bHelp = TRUE;
          PostMessage(hWnd, WM_KEYDOWN, VK_RETURN, 0L);
      }
      break;

  case WM_COMMAND:
      /* Was F1 just pressed in a menu, or are we in help
         mode (Shift+F1)? */

      if (bHelp) {
          dwHelpContextId =
              (wParam == IDM_NEW)?(DWORD)HELPID_FILE_NEW:
              (wParam == IDM_OPEN)? (DWORD)HELPID_FILE_OPEN:
              (wParam == IDM_SAVE)? (DWORD)HELPID_FILE_SAVE:
              (wParam == IDM_SAVEAS)?
                        (DWORD)HELPID_FILE_SAVE_AS:
              (wParam == IDM_PRINT)?(DWORD)HELPID_FILE_PRINT:
              (wParam == IDM_EXIT)? (DWORD)HELPID_FILE_EXIT:
              (wParam == IDM_UNDO)? (DWORD)HELPID_EDIT_UNDO:
              (wParam == IDM_CUT)?(DWORD)HELPID_EDIT_CUT:
              (wParam == IDM_CLEAR)?(DWORD)HELPID_EDIT_CLEAR:
              (wParam == IDM_COPY)? (DWORD)HELPID_EDIT_COPY:
              (wParam == IDM_PASTE)?(DWORD)HELPID_EDIT_PASTE:
              (DWORD)0L;

          if (!dwHelpContextId)
          {Messagebox( hWnd, "Help not available for Help
Menu item",                          "Help Example", MB_OK
return (DefWindowProc(hWnd, message, wParam, lParam));
          }

          bHelp = FALSE;

WinHelp(hWnd,szHelpFileName,HELP_CONTEXT,dwHelpContextId);
          break;
      }
```

Here is how you would do this in Turbo Pascal:

```
    WM_ENTERIDLE:
      if ((WParam=MSGF_MENU) and ((GetKeyState(VK_F1) and
           $8000)0)) then
      begin
        Help:=true;
        PostMessage(Wnd, WM_KEYDOWN, VK_RETURN, 0);
```

```
    end;

WM_COMMAND:

    { Was F1 just pressed in a menu, or are we in help mode
      Shift+F1?  }

    if Help then
    begin
      case WParam of
        IdmNew: HelpContextId:=HelpIdFileNew;
        IdmOpen: HelpContextId:=HelpIdFileOpen;
        IdmSave: HelpContextId:=HelpIdFileSave;
        IdmSaveAs: HelpContextId:=HelpIdFileSaveAs;
        IdmPrint: HelpContextId:=HelpIdFilePrint;
        IdmExit: HelpContextId:=HelpIdFileExit;
        IdmUndo: HelpContextId:=HelpIdEditUndo;
        IdmCut: HelpContextId:=HelpIdEditCut;
        IdmClear: HelpContextId:=HelpIdEditClear;
        IdmCopy: HelpContextId:=HelpIdEditCopy;
        IdmPaste: HelpContextId:=HelpIdEditPaste;
      else
        HelpContextId:=0;
      end;
      if HelpContextId=0 then
      begin
        MessageBox(Wnd, 'Help not available for Help Menu
                   Item',
                   'Help Example', Mb_Ok);
        MainWndProc:=DefWindowProc(Wnd, Message, WParam,
                                   LParam);
        exit;
      end;
      Help:=false;
      WinHelp(Wnd, TempFile, Help_Context, HelpContextId);
    end
```

Detecting F1 in dialog boxes is somewhat more difficult than in menus. You must install a message filter, using the WH_MSGFILTER option of the **SetWindowsHook** function. Your message filter function responds to WM_KEYDOWN and WM_KEYUP messages for VK_F1 when they are sent to a dialog box, as indicated by the MSGF_DIALOGBOX code. By examining the message structure passed to the filter, you can determine the context of the F1 help—what the dialog box is, and the specific option or item. You should not call **WinHelp** while processing the filtered message, but rather post an application-defined message to your application to call **WinHelp** at the first available opportunity.

Getting Help on an Item Listed on the Help Menu

Sometimes users may want information about a general concept in the application rather than about a particular control or window. In these cases, the application should provide Help for a particular topic that is identified by a key word rather than a context identifier.

For example, if the Help file for your application contains a topic that describes how the keyboard is used, you could place a "Keyboard" item in your Help menu. Then, when the user selects that item, your application would call **WinHelp** and request that topic:

```
case IDM_HELP_KEYBOARD:
   WinHelp (hWnd, lpHelpFile, HELP_KEY, (LPSTR)
"Keyboard");
   break;
```

Accessing Additional Key-Word Tables

Your application may have commands or terms that correspond to terms in a similar, but different, application. Given a keyword, the application can call **WinHelp** and look up topics defined in an alternate key-word table. This Multikey functionality is accessed through the **WinHelp** hook with the *wCommand* parameter set to HELP_MULTIKEY.

You specify the footnote character for the alternate key-word table, and the key word or phrase, via a **MULTIKEYHELP** structure which is passed as the *dwData* parameter in the call to **WinHelp**. This structure is defined in WINDOWS.H as:

```
typedef struct tag MULTIKEYHELP {
    WORD mdSize;
    BYTE mkKeyList;
    BYTE szKeyPhrase[1];
} MULTIKEYHELP;
```

The following table lists the format of the fields of the **MULTIKEYHELP** structure:

Parameter	Format
mkSize	The size of the structure, including the key word (or phrase) and the associated key-table letter.
mkKeyList	A single character which defines the footnote character for the alternate key-word table to be searched.
szKeyPhrase	A null-terminated key word or phrase to be looked up in the alternate key-word table.

The following example illustrates a key-word search for the word "frame" in the alternate key-word table designated with the footnote character "L":

```
MULTIKEYHELP mk;
char szKeyword[]="frame";
mk.mkSize=sizeof(MULTIKEYHELP)+(WORD)lstrlen(szKeyword);
mk.mkKeylist="L";
mk.szKeyphrase=szKeyword;
WinHelp(hWnd,lpHelpfile,HELP_MULTIKEY,(LPSTR)&mk);
```

Canceling Help

The Windows Help application is a shared resource that is available to all Windows applications. In addition, since it is a stand-alone application, the user can execute it like any other application. As a result, your application has limited control over the Help application. While your application cannot directly close the Help application window, your application can inform the Help application that Help is no longer needed. Before closing its main window, your application should call **WinHelp** with the *wCommand* parameter set to HELP_QUIT, as shown in the following example, to inform the Help application that your application will not need it again.

```
case WM_DESTROY:
   WinHelp (hWnd, lpHelpFile, HELP_QUIT, NULL);
```

An application that has called **WinHelp** at some point during its execution must call **WinHelp** with the *wCommand* parameter set to HELP_QUIT before the application exits from WinMain (typically during the WM_DESTROY message processing).

If an application opens more than one Help file, then it must call **WinHelp** to quit help for each file.

If an application or DLL has opened a Help file but no longer wants the associated instance of the Help engine (WINHELP.EXE) to remain active, then the application or DLL should call **WinHelp** with the *wCommand* parameter set to HELP_QUIT to destroy the instance of the Help engine.

Under no circumstances should an application or DLL terminate without calling **WinHelp** for any of the opened Help files. A Help file is opened if any other **WinHelp** call has been previously made using the Help filename.

The Windows Help application does not exit until all windows that have called **WinHelp** have called it with *wCommand* set to HELP_QUIT. If an application fails to do so, then the Help application will continue running after all applications that requested Help have terminated.

Summary

This chapter described how to create a Help project file, build the Help resource file, and program your application to access Help. For more information, see the following:

Topic	Reference
Planning a Help system	*Tools*: Chapter 2, "Planning the Help System"
Writing Help topics	*Tools*: Chapter 3, "Creating the Help Topic Files"

Help Examples and Compiler Error Messages

The first part of this chapter contains several examples of Help source files and their corresponding topics as displayed in Help. Each example shows a topic (or part of a topic) as it appears to the Help writer in the RTF-capable word processor and as it appears to the user in the Help window. You can use these examples as guides when creating your own topic files. The examples should help you predict how a particular topic file created in a word processor will appear to the user.

The second part of this chapter contains a list of Help Compiler error messages. Each message is shown as it appears when the compiler encounters the specific error. A short explanation accompanies each message to aid you in solving the problem in your Help system. Preceding the error message listing is a short description of the error reporting behavior of the Help Compiler. Understanding how the compiler reports and reacts to errors will help you to debug your Help files.

Help Topic Examples

Figure 5.1
Word for Windows topic

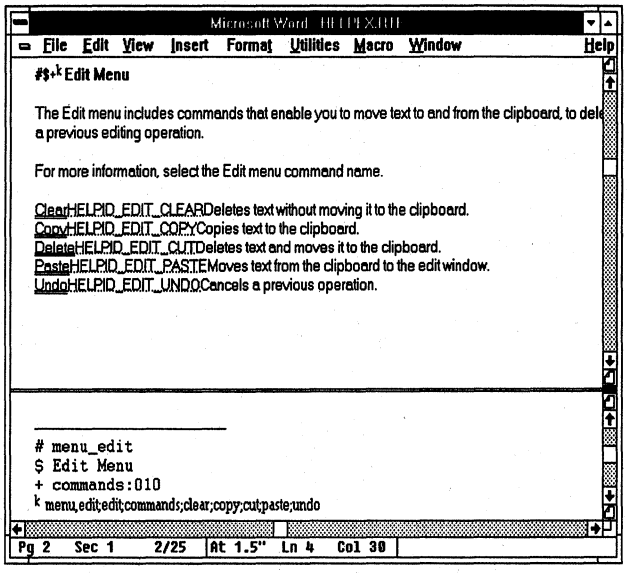

File Edit View Insert Format Utilities Macro Window Help
#$+K Edit Menu
The Edit menu includes commands that enable you to move text to and from the clipboard, to del
a previous editing operation.
For more information, select the Edit menu command name.
ClearHELPID_EDIT_CLEARDeletes text without moving it to the clipboard.
CopyHELPID_EDIT_COPYCopies text to the clipboard.
DeleteHELPID_EDIT_CUTDeletes text and moves it to the clipboard.
PasteHELPID_EDIT_PASTEMoves text from the clipboard to the edit window.
UndoHELPID_EDIT_UNDOCancels a previous operation.
menu_edit
$ Edit Menu
+ commands:010
K menu,edit;edit;commands;clear;copy;cut;paste;undo
Pg 2 Sec 1 2/25 At 1.5" Ln 4 Col 30

Figure 5.2
Help topic display

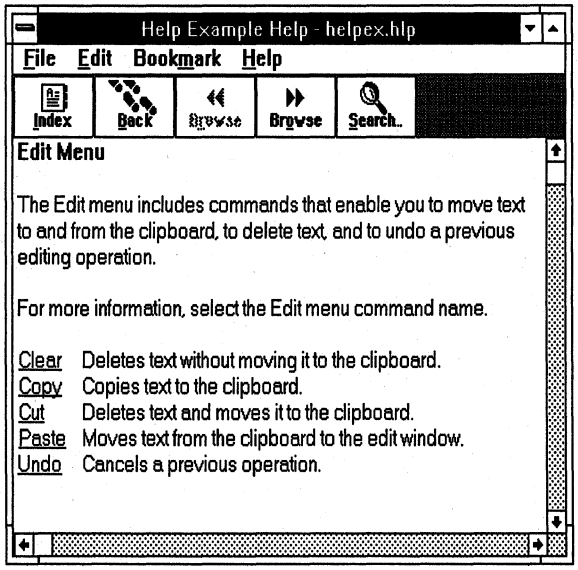

Help Example Help - helpex.hlp
File Edit Bookmark Help
Index
Back
Browse
Browse
Search..
Edit Menu
The Edit menu includes commands that enable you to move text to and from the clipboard, to delete text, and to undo a previous editing operation.
For more information, select the Edit menu command name.
Clear Deletes text without moving it to the clipboard.
Copy Copies text to the clipboard.
Cut Deletes text and moves it to the clipboard.
Paste Moves text from the clipboard to the edit window.
Undo Cancels a previous operation.

Figure 5.3
Mac® Word topics

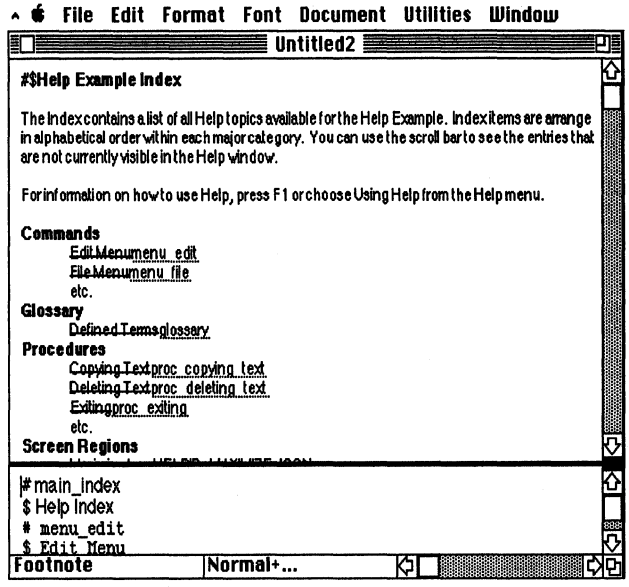

File Edit Format Font Document Utilities Window
Untitled2
#$Help Example Index
The Index contains a list of all Help topics available for the Help Example. Index items are arrange in alphabetical order within each major category. You can use the scroll bar to see the entries that are not currently visible in the Help window.
For information on how to use Help, press F1 or choose Using Help from the Help menu.
Commands
Edit Menumenu_edit
File Menumenu_file
etc.
Glossary
Defined Termsglossary
Procedures
Copying Textproc_copying_text
Deleting Textproc_deleting_text
Exitingproc_exiting
etc.
Screen Regions
#main_index
$Help Index
#menu_edit
$Edit Menu
Footnote
Normal+...

Figure 5.4
Help topic display

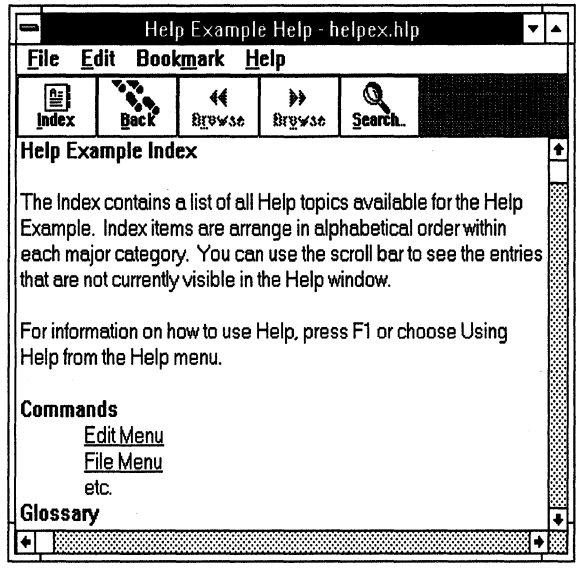

Help Example Help - helpex.hlp
File Edit Bookmark Help
Index
Back
Browse
Browse
Search...
Help Example Index
The Index contains a list of all Help topics available for the Help Example. Index items are arrange in alphabetical order within each major category. You can use the scroll bar to see the entries that are not currently visible in the Help window.
For information on how to use Help, press F1 or choose Using Help from the Help menu.
Commands
Edit Menu
File Menu
etc.
Glossary

Figure 5.5
Word 5.0 for DOS topic

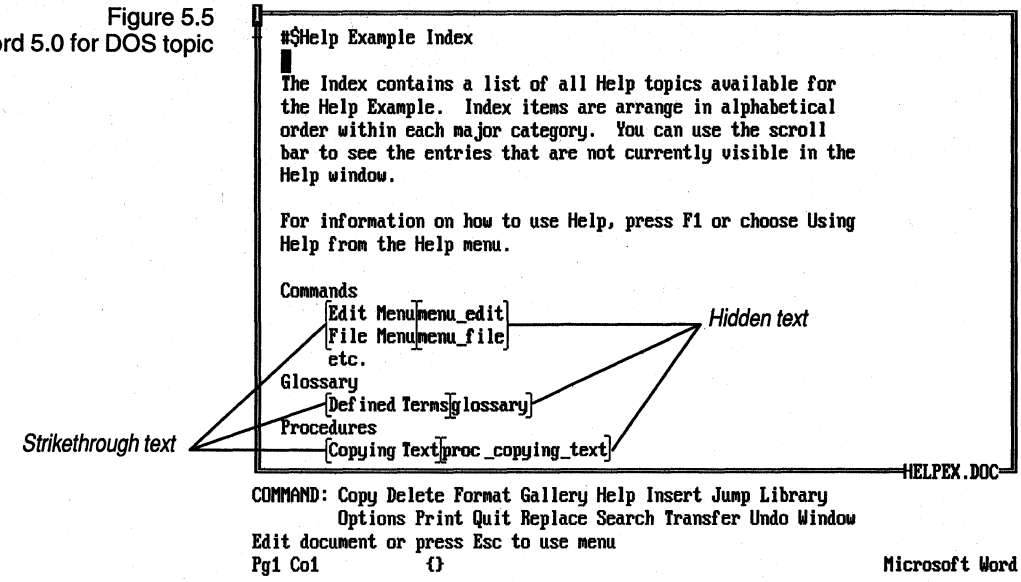

#$Help Example Index
The Index contains a list of all Help topics available for the Help Example. Index items are arrange in alphabetical order within each major category. You can use the scroll bar to see the entries that are not currently visible in the Help window.
For information on how to use Help, press F1 or choose Using Help from the Help menu.
Commands
Edit Menu menu_edit
File Menu menu_file
etc.
Glossary
Defined Terms glossary
Procedures
Copying Text proc _copying_text
Hidden text
Strikethrough text
HELPEX.DOC
COMMAND: Copy Delete Format Gallery Help Insert Jump Library
Options Print Quit Replace Search Transfer Undo Window
Edit document or press Esc to use menu
Pg1 Co1
Microsoft Word

Figure 5.6
Help topic display

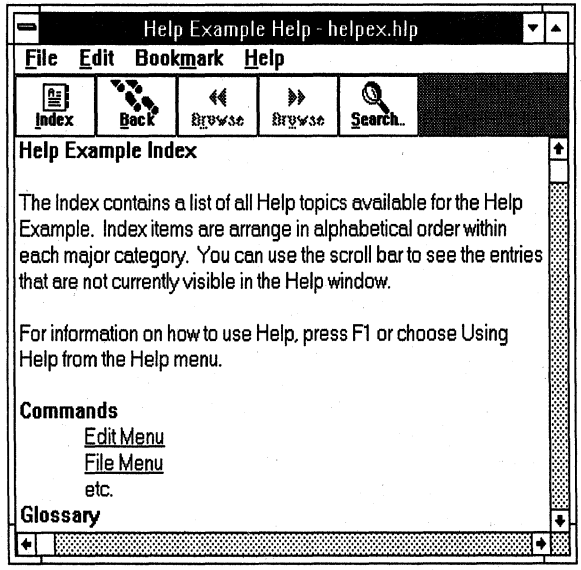

Help Example Help - helpex.hlp
File Edit Bookmark Help
Index
Back
Browse
Browse
Search..
Help Example Index
The Index contains a list of all Help topics available for the Help Example. Index items are arrange in alphabetical order within each major category. You can use the scroll bar to see the entries that are not currently visible in the Help window.
For information on how to use Help, press F1 or choose Using Help from the Help menu.
Commands
Edit Menu
File Menu
etc.
Glossary

Figure 5.7
Bitmap by reference in topic

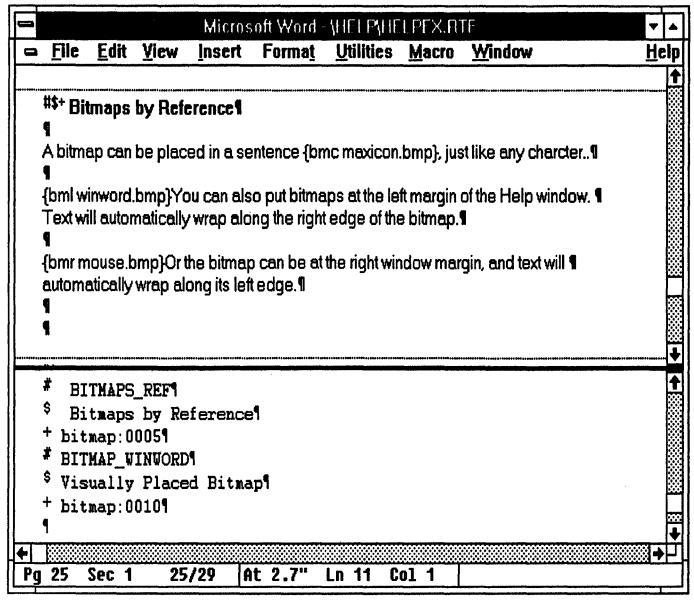

Microsoft Word - HELPWHELPEX.RTF
File Edit View Insert Format Utilities Macro Window Help
#$+ Bitmaps by Reference¶
¶
A bitmap can be placed in a sentence {bmc maxicon.bmp}, just like any charcter..¶
¶
{bml winword.bmp}You can also put bitmaps at the left margin of the Help window. ¶
Text will automatically wrap along the right edge of the bitmap.¶
¶
{bmr mouse.bmp}Or the bitmap can be at the right window margin, and text will ¶
automatically wrap along its left edge.¶
¶
¶
BITMAPS_REF¶
$ Bitmaps by Reference¶
+ bitmap:0005¶
# BITMAP_WINWORD¶
$ Visually Placed Bitmap¶
+ bitmap:0010¶
¶
Pg 25 Sec 1 25/29 At 2.7" Ln 11 Col 1

Figure 5.8
Help topic display

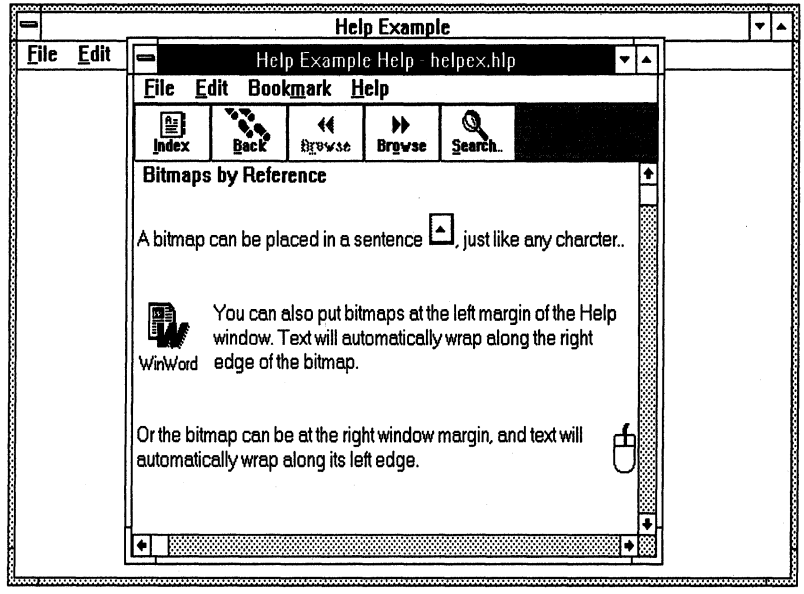

Help Example
File Edit
Help Example Help - helpex.hlp
File Edit Bookmark Help
Index
Back
Browse
Browse
Search..
Bitmaps by Reference
A bitmap can be placed in a sentence , just like any charcter..
WinWord
You can also put bitmaps at the left margin of the Help window. Text will automatically wrap along the right edge of the bitmap.
Or the bitmap can be at the right window margin, and text will automatically wrap along its left edge.

Figure 5.9
Visual bitmap in topic

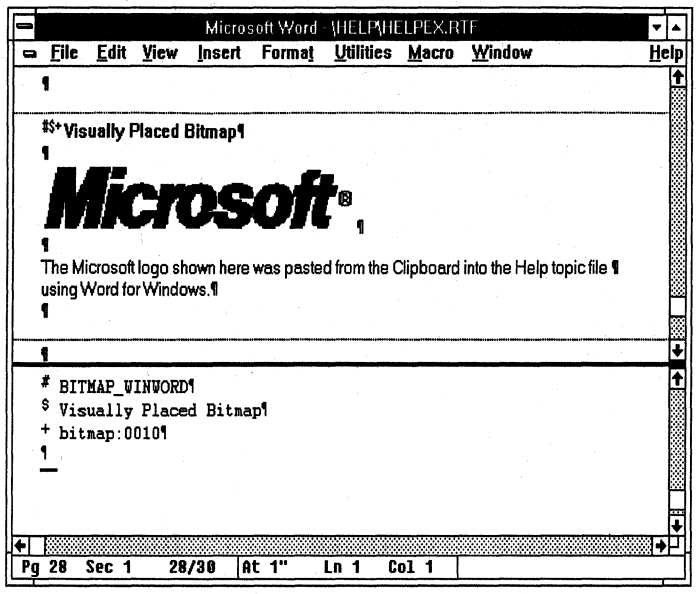
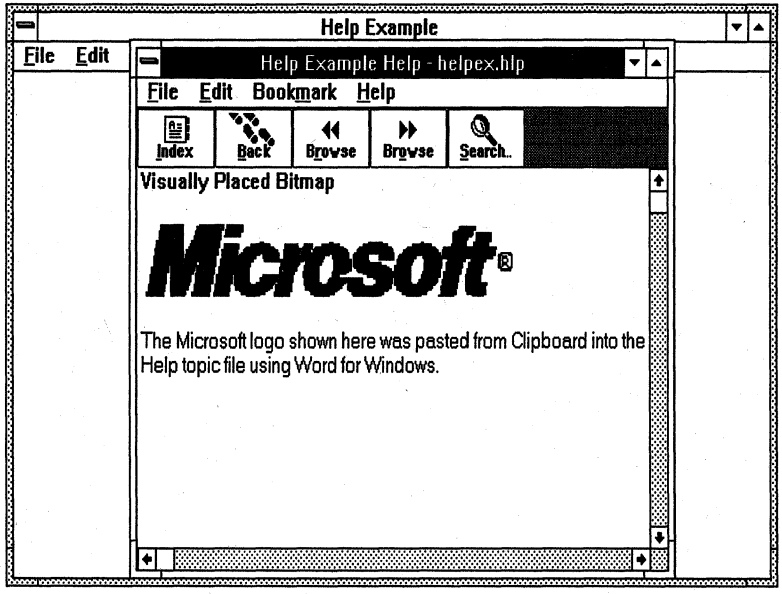

Microsoft Word - \HELP\HELPEX.RTF
File Edit View Insert Format Utilities Macro Window Help
$* Visually Placed Bitmap
Microsoft®
The Microsoft logo shown here was pasted from the Clipboard into the Help topic file
using Word for Windows.
BITMAP_WINWORD
$ Visually Placed Bitmap
+ bitmap:0010
Pg 28 Sec 1 28/30 At 1" Ln 1 Col 1

Figure 5.10
Help topic display



The following is the Helpex (sample Help) project file:

```
[OPTIONS]
ROOT=c:\help
INDEX=main_index
TITLE=Help Example
COMPRESS=true

[FILES]
helpex.rtf  ; jump topics
terms.rtf   ; look-up terms

[MAP]
main_index 0xFFFF
#define HELPID_EDIT_CLEAR     100
#define HELPID_EDIT_COPY      101
#define HELPID_EDIT_CUT       102
#define HELPID_EDIT_PASTE     103
#define HELPID_EDIT_UNDO      104
#define HELPID_FILE_EXIT      200
#define HELPID_FILE_NEW       201
#define HELPID_FILE_OPEN      202
#define HELPID_FILE_PRINT     203
#define HELPID_FILE_SAVE      204
#define HELPID_FILE_SAVE_AS   205
#define HELPID_EDIT_WINDOW    300
#define HELPID_MAXIMIZE_ICON  301
#define HELPID_MINIMIZE_ICON  302
#define HELPID_SPLIT_BAR      303
#define HELPID_SIZE_BOX       304
#define HELPID_SYSTEM_MENU    305
#define HELPID_TITLE_BAR      306
#define HELPID_SIZING_BORDER  307
```

Help Compiler Error Messages

The Help Compiler displays messages when it encounters errors in building the Help resource file. Errors during processing of the project file are numbered beginning with the letter *P* and appear as in the following examples:

```
Error P1025: line...7 of filename.HPJ : Section heading
sectionname unrecognized.

Warning P1039: line...38 of filename.HPJ : [BUILDTAGS]
section missing.
```

Errors which occur during processing of the RTF topic file(s) are numbered beginning with the letter *R* and appear as in the following examples:

```
Error R2025: File environment error.

Warning R2501: Using old key-phrase table.
```

Whenever possible, the compiler will display the topic number and/or filename that contains the error. Though topics are not numbered, the topic number given with an error message refers to that topic's sequential position in your RTF file (first, second, etc.). These numbers may be identical to the page number shown by your word processor, depending on the number of lines you have assigned to the hypothetical printed page. Remember that topics are separated by hard page breaks, even though there is no such thing as a "page" in the Help system.

Messages beginning with the word "Error" are fatal errors. Errors are always reported, and no usable Help resource file will result from the build. Messages beginning with the word "Warning" are less serious in nature. A build with warnings will produce a valid Help resource file which will load under Windows, but the file may contain operational errors. You can specify the amount of warning information to be reported by the compiler. See the section in Chapter 3 titled "Specifying Error Reporting: The Warning Option" for more information on choosing warning levels to be displayed.

The compiler's reaction to an error is described for each error in the listing that follows. During processing of the project file, the compiler ignores lines that contain errors and attempts to continue with the build. This means that errors encountered early in a build may result in many more errors being reported as the build continues. Similarly, errors during processing of the RTF topic files will be reported and if not serious, the compiler will continue with the build. A single error condition in the topic file may result in more than one error message being reported by the compiler. For instance, a misidentified topic will cause an error to be reported every time jump terms refer to the correct topic identifier. Such a mistake is easily rectified by simply correcting the footnote containing the wrong context string.

Errors During Processing of Project File

P1001 **Unable to read file *filename*.**
The file specified in the project file is unreadable. This is a DOS file error.

P1003 **Invalid path specified in Root option.**
The path specified by the Root option cannot be found. The compiler uses the current working directory.

P1005 **Path and filename exceed limit of 79 characters.**
The absolute pathname, or the combined root and relative pathname, exceed the DOS limit of 79 characters. The file is skipped.

P1007 **Root path exceeds maximum limit of 66 characters.**
The specified root pathname exceeds the DOS limit of 66 characters. The pathname is ignored and the compiler uses the current working directory.

P1009 **[FILES] section missing.**
The [Files] section is required. The compilation is aborted.

P1011 **Option *optionname* previously defined.**
The specified option was defined previously. The compiler ignores the attempted redefinition.

P1013 **Project file extension cannot be .HLP.**
You cannot specify that the compiler use a project file with the .HLP extension. Normally, project files are given the .HPJ extension.

P1015 **Unexpected end-of-file.**
The compiler has unexpectedly come to the end of the project file. There might be an open comment in the project file or an included file.

P1017 **Parameter exceeds maximum length of 128 characters.**
An option, context name or number, build tag, or other parameter on the specified line exceeds the limit of 128 characters. The line is ignored.

P1021 **Context number already used in [MAP] section.**
The context number on the specified line in the project file was previously mapped to a different context string. The line is ignored.

P1023 **Include statements nested too deeply.**
The **#include** statement on the specified line has exceeded the maximum of five include levels.

P1025 **Section heading *sectionname* unrecognized.**
A section heading that is not supported by the compiler has been used. The line is skipped.

P1027 **Bracket missing from section heading *sectionname*.**
The right bracket (]) is missing from the specified section heading. Insert the bracket and compile again.

P1029 **Section heading missing.**
The section heading on the specified line is not complete. This error is also reported if the first entry in the project file is not a section heading. The compiler continues with the next line.

P1030 **Section *sectionname* previously defined.**
A duplicate section has been found in the project file. The lines under the duplicated section heading are ignored and the compiler continues from the next valid section heading.

P1031 **Maximum number of build tags exceeded.**
The maximum number of build tags that can be defined is 30. The excess tags are ignored.

P1033 **Duplicate build tag in [BUILDTAGS] section.**
A build tag in the [BuildTags] section has been repeated unnecessarily

P1035 **Build tag length exceeds maximum.**
The build tag on the specified line exceeds the maximum of 32 characters. The compiler skips this entry.

P1037 **Build tag *tagname* contains invalid characters.**
Build tags can contain only alphanumeric characters or the underscore (_) character. The line is skipped.

P1039 **[BUILDTAGS] section missing.**
The **BUILD** option declared a conditional build, but there is no [BuildTags] section in the project file. All topics are included in the build.

P1043 **Too many tags in Build expression.**
The Build expression on the specified line has used more than the maximum of 20 build tags. The compiler ignores the line.

P1045 **[ALIAS] section found after [MAP] section.**
When used, the [Alias] section must precede the [Map] section in the project file. The [Alias] section is skipped otherwise.

P1047 **Context string *contextname* already assigned an alias.**
You cannot do: a=b then a=c (A context string can only have one alias.)

The specified context string has previously been aliased in the [Alias] section. The attempted reassignment on this line is ignored.

P1049 **Alias string aliasname already assigned.**
You cannot do: a=b then b=c (You can't alias an alias.)

An alias string cannot, in turn, be assigned another alias.

P1051 **Context string *contextname* cannot be used as alias string.**
You cannot do: a=b then c=a

A context string that has been assigned an alias cannot be used later as an alias for another context string.

P1053 **Maximum number of font ranges exceeded.**
The maximum number of font ranges that can be specified is five. The rest are ignored.

P1055 **Current font range overlaps previously defined range.**
A font size range overlaps a previously defined mapping. Adjust either font range to remove any overlaps. The second mapping is ignored.

P1056 **Unrecognized font name in Forcefont option.**
A font name not supported by the compiler has been encountered. The font name is ignored and the compiler uses the default Helvetica font.

P1057 **Font name too long.**
Font names cannot exceed 20 characters. The font is ignored.

P1059 **Invalid multiple-key syntax.**
The syntax used with a **MULTIKEY** option is unrecognized. See Chapter 4, "Building the Help File," for the proper syntax.

P1061 **Character already used.**
The specified keyword–table identifier is already in use. Choose another character.

P1063 **Characters 'K' and 'k' cannot be used.**
These characters are reserved for Help's normal key-word table. Choose another character.

P1065 **Maximum number of keyword tables exceeded.**
The limit of five keyword tables has been exceeded. Reduce the number. The excess tables are ignored.

P1067 **Equal sign missing.**
An option is missing its required equal sign on the specified line. Check the syntax for the option.

P1069 **Context string missing.**
The line specified is missing a context string before an equal sign.

P1071 **Incomplete line in *sectionname* section.**
The entry on the specified line is not complete. The line is skipped.

P1073 **Unrecognized option in [OPTIONS] section.**
An option has been used that is not supported by the compiler. The line is skipped.

P1075 **Invalid build expression.**
The syntax used in the build expression on the specified line contains one or more logical or syntax errors.

P1077 **Warning level must be 1, 2, or 3.**
The **WARNING** reporting level can only be set to 1, 2, or 3. The compiler will default to full reporting (level 3).

P1079 **Invalid compression option.**
The **COMPRESS** option can only be set to TRUE or FALSE. The compilation continues without compression.

P1081 **Invalid title string.**
The **TITLE** option defines a string that is null or contains more than 32 characters. The title is truncated.

P1083 **Invalid context identification number.**
The context number on the specified line is null or contains invalid characters.

P1085 **Unrecognized text.**
The unrecognizable text that follows valid text in the specified line is ignored.

P1086 **Invalid font-range syntax.**
The font-range definition on the specified line contains invalid syntax. The compiler ignores the line. Check the syntax for the **MAPFONTSIZE** option.

P1089 **Unrecognized sort ordering.**
You have specified an ordering that is not supported by the compiler. Contact Borland Technical Support for clarification of the error.

R2001 **Unable to open bitmap file** *filename.*
The specified bitmap file is unreadable. This is a DOS file error.

R2003 **Unable to include bitmap file** *filename.*
The specified bitmap file could not be found or is unreadable. This is a DOS file error or an out-of-memory condition.

R2005 **Disk full.**
The Help resource file could not be written to disk. Create more space on the destination drive.

R2009 **Cannot use reserved DOS device name for file** *filename.*
A file has been referred to as COM1, LPT2, PRN, etc. Rename the file.

R2013 **Output file** *filename* **already exists as a directory.**
There is a subdirectory in the Help project root with the same name as the desired Help resource file. Move or rename the subdirectory.

R2015 **Output file** *filename* **already exists as read-only.**
The specified filename cannot be overwritten by the Help resource file because the file has a read-only attribute. Rename the project file or change the file's attribute.

R2017 **Path for file** *filename* **exceeds limit of 79 characters.**
The absolute pathname, or the combined root and relative pathname, to the specified file exceed the DOS limit of 79 characters. The file is ignored.

R2019 **Cannot open file** *filename.*
The specified file is unreadable. This is a DOS file error.

R2021 **Cannot find file** *filename.*
The specified file could not be found or is unreadable. This is a DOS file error or an out-of-memory condition.

R2023 **Not enough memory to build Help file.**
To free up memory, unload any unneeded applications, device drivers, and memory-resident programs.

R2025 **File environment error.**
The compiler has insufficient available file handles to continue. Increase the values for FILES= and BUFFERS= in your CONFIG.SYS file and reboot.

R2027 **Build tag *tagname* not defined in [BUILDTAGS] section of project file.**
The specified build tag has been assigned to a topic, but not declared in the project file. The tag is ignored for the topic.

R2033 **Context string in Map section not defined in any topic.**
There are one or more context strings defined in the project file that the compiler could not find topics for.

R2035 **Build expression missing from project file.**
The topics have build tags, but there is no Build= expression in the project file. The compiler includes all topics in the build.

R2037 **File *filename* cannot be created, due to previous error(s).**
The Help resource file could not be created because the compiler has no topics remaining to be processed. Correct the errors that preceded this error and recompile.

R2039 **Unrecognized table formatting in topic *topicnumber* of file *filename*.**
The compiler ignores table formatting that is unsupported in Help. Reformat the entries as linear text if possible.

R2041 **Jump *context_string* unresolved in topic *topicnumber* of file *filename*.**
The specified topic contains a context string that identifies a nonexistent topic. Check spelling, and that the desired topic is included in the build.

R2043 **Hotspot text cannot spread over paragraphs.**
A jump term spans two paragraphs. Remove the formatting from the paragraph mark.

R2045 **Maximum number of tab stops reached in topic *topicnumber* of file *filename*.**
The limit of 32 tab stops has been exceeded in the specified topic. The default stops are used after the 32nd tab.

R2047 **File *filename* not created.**
There are no topics to compile, or the build expression is false for all topics. There is no Help resource file created.

R2049 **Context string text too long in topic *topicnumber* of file *filename*.**
Context string hidden text cannot exceed 64 characters. The string is ignored.

R2051 **File *filename* is not a valid RTF topic file.**
The specified file is not an RTF file. Check that you have saved the topic as RTF from your word processor.

R2053 **Font *fontname* in file *filename* not in RTF font table.**
A font not defined in the RTF header has been entered into the topic. The compiler uses the default system font.

R2055 **File *filename* is not a usable RTF topic file.**
The specified file contains a valid RTF header, but the content is not RTF or is corrupted.

R2057 **Unrecognized graphic format in topic *topicnumber* of file *filename*.**
The compiler supports only Windows bitmaps. Check that metafiles or Macintosh formats have not been used. The graphic is ignored.

R2059 **Context string identifier already defined in topic *topicnumber* of file *filename*.**
There is more than one context-string identifier footnote for the specified topic. The compiler uses the identifier defined in the first # footnote.

R2061 **Context string *contextname* already used in file *filename*.**
The specified context string was previously assigned to another topic. The compiler ignores the latter string and the topic has no identifier.

R2063 **Invalid context-string identifier for topic *topicnumber* of file *filename*.**
The context-string footnote contains nonalphanumeric characters or is null. The topic is not assigned an identifier.

R2065 **Context string defined for index topic is unresolved.**
The index topic defined in the project file could not be found. The compiler uses the first topic in the build as the index.

R2067 **Footnote text too long in topic *topicnumber* of file *filename*.**
Footnote text cannot exceed the limit of 1000 characters. The footnote is ignored.

R2069 **Build tag footnote not at beginning of topic *topicnumber* of file *filename*.**
The specified topic contains a buildtag footnote that is not the first character in the topic. The topic is not assigned a build tag.

R2071 **Footnote text missing in topic *topicnumber* of file *filename*.**
The specified topic contains a footnote that has no characters.

R2073 **Keyword string is null in topic *topicnumber* of file *filename*.**
A key-word footnote exists for the specified topic, but contains no characters.

R2075 **Keyword string too long in topic** *topicnumber* **of file** *filename.*
The text in the key-word footnote in the specified topic exceeds the limit of 255 characters. The excess characters are ignored.

R2077 **Keyword(s) defined without title in topic** *topicnumber* **of file** *filename.*
Key word(s) have been defined for the specified topic, but the topic has no title assigned. Search Topics Found displays Untitled Topic< for the topic.

R2079 **Browse sequence string is null in topic** *topicnumber* **of file** *filename.*
The browse-sequence footnote for the specified topic contains no sequence characters.

R2081 **Browse sequence string too long in topic** *topicnumber* **of file** *filename.*
The browse-sequence footnote for the specified topic exceeds the limit of 128 characters. The sequence is ignored.

R2083 **Missing sequence number in topic** *topicnumber* **of file** *filename.*
A browse-sequence number ends in a colon (:) for the specified topic. Remove the colon, or enter a "minor" sequence number.

R2085 **Sequence number already defined in topic** *topicnumber* **of file** *filename.*
There is already a browse-sequence footnote for the specified topic. The latter sequence is ignored.

R2087 **Build tag too long.**
A build tag for the specified topic exceeds the maximum of 32 characters. The tag is ignored for the topic.

R2089 **Title string null in topic** *topicnumber* **of file** *filename.*
The title footnote for the specified topic contains no characters. The topic is not assigned a title.

R2091 **Title too long in topic** *topicnumber* **of file** *filename.*
The title for the specified topic exceeds the limit of 128 characters. The excess characters are ignored.

R2093 **Title titlename in topic** *topicnumber* **of file** *filename* **used previously.**
The specified title has previously been assigned to another topic.

R2095 **Title defined more than once in topic** *topicnumber* **of file** *filename.*
There is more than one title footnote in the specified topic. The compiler uses the first title string.

R2501 **Using old key-phrase table.**
Maximum compression can only result by deleting the .PH file before each recompilation of the Help topics.

R2503 **Out of memory during text compression.**
The compiler encountered a memory limitation during compression. Compilation continues with the Help resource file not compressed. Unload any unneeded applications, device drivers, and memory-resident programs.

R2505 **File environment error during text compression.**
The compiler has insufficient available file handles for compression. Compilation continues with the Help resource file not compressed. Increase the values for FILES= and BUFFERS= in your CONFIG.SYS file and reboot.

R2507 **DOS file error during text compression.**
The compiler encountered a problem accessing a disk file during compression. Compilation continues with the Help resource file not compressed.

R2509 **Error during text compression.**
One of the three compression errors—R2503, R2505, or R2507—has occurred. Compilation continues with the Help resource file not compressed.

R2701 **Internal error.**
Contact Borland Technical Support for clarification of the error.

R2703 **Internal error.**
Contact Borland Technical Support for clarification of the error.

R2705 **Internal error.**
Contact Borland Technical Support for clarification of the error.

R2707 **Internal error.**
Contact Borland Technical Support for clarification of the error.

R2709 **Internal error.**
Contact Borland Technical Support for clarification of the error.

Index

Help
  audience definition 8
  cancelling 67-68
  compiler error messages 76-86
  context-sensitive 11, 12, 60, 62, 64, 65
  control codes 25
  error messages 76-86
  F1 support 63-65
  file structure 13-15
  file. See Help Project file 43
  key words 31
  keyword table, accessing 66, 67
  keywords 30-32
  on Help menu item 66
  planning
    overview 7
  topic files
    examples 69-76
    links summarized 6
Help compiler 56, 57
Help graphics 20
  bitmaps
    creating, capturing 37
    placing 38, 39
Help index 9
Help Project file
  accessing from an application 58
  Alias section 53, 54
  Bitmaps section 56
  bitmaps, including by reference 56
  Build option 47, 48
  Build Tags section 45
  compiling 56, 57
  Compress option 52, 53
  context strings, alternate 53, 54
  context-sensitive Help 60, 62, 64, 65
  context-sensitive topics 54, 55
  creating 43, 44
  F1 support 63, 64, 65
  Files section 44, 45
  Forcefont option 50, 51
  Index option 49
  keyword table, accessing 66, 67
  Map section 54, 55
  Mapfontsize option 51
  Multikey option 51, 52
  on Help menu item 66
  Options section 46
  Root option 48, 49
  Title option 50
  Warning option 46, 47
Help system
  appearance to programmer 6
  appearance to user 4
  appearance to writer 5
  calling WinHelp 58, 59, 60
  development cycle described 1, 2, 3
  graphics. See Help graphics 20
  See also Help
  hypertext links summarized 6
  topics. See Help topics 8
Help text
  fonts 18, 19
  layout 16, 17, 18
Help topic files
  authoring tool 23
  browse sequence numbers 32-34
  build tags 26-28
  context strings 28, 29
  control codes 25
  cross references 34
  definitions 35, 36
  graphics 37
  jumps 34
  keywords 30, 31, 32
  managing 39
  title footnotes 29, 30
  tracking 40, 41

Help topics
content 8
context-sensitivity 11, 12
cross-references 34
definitions 35, 36
file structure 13-15
files. See Help topic files 24
jumps 34
structure of 9, 10
text. See Help text 16
Help Tracker 40, 41

BORLAND LANGUAGES HELP COMPILER

BORLAND

CORPORATE HEADQUARTERS: 100 BORLAND WAY, SCOTTS VALLEY, CA 95066-3249, (408) 431-1000. OFFICES IN: AUSTRALIA, BELGIUM, CANADA, DENMARK, FRANCE, GERMANY, HONG KONG, ITALY, JAPAN, KOREA, LATIN AMERICA, MALAYSIA, NETHERLANDS, NEW ZEALAND, SINGAPORE, SPAIN, SWEDEN, TAIWAN, AND UNITED KINGDOM ■ PART #AAMN-LAN01 ■ BOR 2279